Exploring the Ecosystem of Referrer-Anonymizing Services

Nick Nikiforakis, Steven Van Acker, Frank Piessens, and Wouter Joosen

IBBT-DistriNet, KU Leuven, 3001 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Abstract. The constant expansion of the World Wide Web allows users to enjoy a wide range of products and services delivered directly to their browsers. At the same time however, this expansion of functionality is usually coupled with more ways of attacking a user's security and privacy. In this arms race, certain web-services present themselves as privacy-preserving or privacy-enhancing. One type of such services is a Referrer-Anonymizing Service (RAS), a service which relays users from a source site to a destination site while scrubbing the contents of the referrer header from user requests.
In this paper, we investigate the ecosystem of RASs and how they interact with web-site administrators and visiting users. We discuss their workings, what happens behind the scenes and how top Internet sites react to traffic relayed through such services. In addition, we present user statistics from our own Referrer-Anonymizing Service and show the leakage of private information by others towards advertising agencies as well as towards 'curious' RAS owners.

Keywords: referrer, anonymization, online ecosystem

1 Introduction

In the infant stages of the Internet, privacy and anonymity were mostly unnecessary due to the small size of the online community and the public nature of the available data. Today however, this has changed. People have online identities, are connected to the Internet almost permanently and they increasingly store their sensitive documents, photos and other data online in the cloud. Our new online way of life provides interesting opportunities to those who seek to exploit it. In an extreme case, corrupt regimes trying to find out what their citizens are doing and thinking, want to violate both online privacy and anonymity [9]. The threat however, need not be a far-fetched scenario or exclusive to the paranoid: large companies and organizations are also interested in the online habits of the masses for various reasons, e.g. targeted advertising.

Projects like The Onion Router (TOR) [11, 25] and the Invisible Internet Project (I2P) [16] provide online anonymity to their users by routing Internet traffic through a number of relays, thus making it harder for the endpoint to trace

the source of the traffic. The application-layer however, on top of the network-layer where TOR or I2P reside, could still carry information that can compromise a user's anonymity or privacy. This is especially so when a web-browser is used, because browsers leak a wealth of information about their users. A study [13] by the EFF's Panopticlick project [1] shows that, based on data typically provided by a browser, a web-site visitor can be uniquely identified in the majority of cases. Private details can be extracted even in the cases where users utilize their browsers' private modes [2] or spoof their user-agent information [15].

One particularly sensitive piece of data, transmitted with almost every HTTP request but commonly overlooked, is the referrer information in the 'Referer' [1] header, which can be used to trace the page where a visitor came from. Online services known as Referrer-Anonymizing Services (RASs) scrub this referrer information from HTTP requests, providing both anonymity to web-sites hosting links as well as privacy to users following those links. In this paper, we take a closer look at RASs. We first perform a manual analysis of popular RASs and record their workings and architectural choices. Through a series of experiments we approach RASs from three different perspectives: the perspective of sites utilizing RASs, the RASs themselves, and the destination sites receiving traffic relayed through a RAS. In the first experiment, we determine what type of sites make use of a RAS and for what reason. The second experiment analyzes the data that RASs have access to and whether they actually protect the privacy of visitors and the anonymity of linking sites. In the last experiment, we observe the reactions of popular web-sites when they are exposed to incoming links relayed through a RAS. From these experiments, we can conclude that in several cases, user privacy is sacrificed for the linking site's anonymity and that not all RASs can be trusted with private data.

The main contributions of this paper are:

- Large-scale study of RASs and their common features
- Experimental evidence of privacy and anonymity violations from RASs
- Identification of types of RAS users and the rationale behind their usage
- Analysis of third-party site responses towards traffic relayed through RASs showing that RAS-relayed traffic is occasionally not well-received

The rest of the paper is structured as follows: In Section 2 we provide background information about referrers and how they can be used but also abused. In Section 3 we describe how RASs work in general followed by a taxonomy of 30 real-world RASs in Section 4. In Section 5 we study how well these 30 RASs protect the anonymity and privacy of their users. In Section 6, we categorize users of RASs and their purpose based on data gathered by our own RAS. In Section 7, we expose popular Internet sites to requests anonymized by RASs and measure their behavior. We discuss related work in Section 8 and we conclude in Section 9.

[1] The correct spelling is 'referrer'. The misspelled word 'referer' was introduced by mistake by Phillip Hallam-Baker [17] and later adopted into the HTTP specification.

2 Background

In this section we briefly go over the workings of the referrer header and we list some valid use-cases as well as abuse-cases for this header.

2.1 Referrer Header

In the HTTP protocol, all client-side requests and server-side responses have headers and optionally a data body. At the client-side, each request contains headers that, at minimum, ask for a specific resource from the web-server, in a `GET` or `POST` manner and in the context of a specific web-site (`Host`), since typically a single web-server serves more than just one web-site. On top of these headers, browsers add a wide range of other headers, the most common of which are headers specifying the user's cookies towards a specific web-site, the user-agent and the encoding-schemes accepted by the current browser.

An HTTP header that is less known but as present in requests as all aforementioned headers, is the '`Referer`'. The HTTP referrer header is automatically added by the browser to outgoing requests, and identifies the URI of the resource from which the current request originated [24]. For instance, if a user while being on `www.example.com/index.php?id=42`, clicks on a link to `www.shopping.com`, her browser would emit a request similar to the following one:

```
GET / HTTP/1.1
Host: www.shopping.com
User-Agent: Mozilla/5.0 (X11; Linux i686)
Accept: text/html,application/xhtml+xml
Proxy-Connection: keep-alive
Referer: http://www.example.com/index.php?p=42
```

In this request, the user's browser provides to `www.shopping.com` the exact location of the page containing the clicked link, resulting in the request towards their servers. This behavior is true not only when a user clicks on a link, but also on all the non-voluntary requests that a browser automatically initiates while parsing a page. For example, all requests created while fetching remote images, scripts, cascading style sheets and embedded objects will contain the referrer header. The referrer header is traditionally omitted in one of the following cases:*(i)* when users manually type a URI in their browser's address bar, *(ii)* when users click on an existing browser bookmark and *(iii)* when users are on a HTTPS site and click on an HTTP link.

In HTML5, the web-programmer can add a special 'noreferrer' attribute to selected anchor link tags that will cause the browser not to emit the referrer header when these links are clicked [26]. At the time of this writing, from the most popular three browsers (Mozilla Firefox, Google Chrome and Internet Explorer), Google Chrome is the only browser which supports this new 'noreferrer' attribute. We believe that this lack of browser support will only amplify the hesitation of web-developers in trying and adopting new security/privacy mechanisms [31]. For this reason, we do not expect widespread use of this 'noreferrer' attribute any time in the near future.

2.2 Referrer Use-cases

In this section we provide a non-exhaustive list of legitimate uses of the HTTP referrer for the web-server which receives the referrer-containing request:

- **Advertising programs**: In many cases, a web-site will buy banner/link placement space on more than one third-party web-sites. Using the referrer header, the advertised site can assess the percentage of visitors coming from each third-party site and use this information to either renew or cancel its advertising contracts.
- **CSRF Protection**: Cross-site Request Forgery (CSRF) is a type of attack where the attacker abuses the established trust between a web-site and a browser [30]. In the typical scenario, a victim who has an active session cookie with a web-application is lured into visiting a malicious site which initiates arbitrary requests towards that web-application in the background. The victim's browser appends the session cookie to each request thus validating them towards the web-server. Due to the way this attack is conducted, the referrer header in the malicious requests will not be the same as when the requests are conducted by the user, from within the web-application. Thus, a simple countermeasure against CSRF attacks is to allow the requests containing the expected referrer header and deny the rest.
- **Deep-linking detection**: Deep-linking or 'hotlinking' is the practice of linking directly to an object on a remote site without linking to any other part of the remote site's content. In practice, this behavior is unwanted, when the object that is deep-linked was originally given to users after a series of necessary steps (e.g. giving access to a music file after filling out an online survey) [8]. By checking the referrer header before releasing the object, the site can protect itself from users who did not go through the expected series of steps. Unfortunately this approach can be easily circumvented by users who change the referrer header values of their requests to match the expected value of the remote site, using a modified browser.
- **Access-Control**: Using the same reasoning as in deep-linking, a site can enforce access control to individual pages by making sure that the visiting user arrives there only from other selected destinations. This technique is also used to provide special offers when a site is visited from another site that would normally not be visible to regular users. Wondracek et al. [28] discovered that this technique is used by traffic brokers in adult-content web-sites. As in the previous use case, this sort of access-control can be bypassed by a user with malicious intent.
- **Statistics gathering**: In large and complex web-sites, the web-developers seek to understand whether the content layout is facilitating the user into finding the data that they need. Through the use of the referrer, web-applications can track users between pages (without the need for cookies) and find the most common visitor paths.

2.3 Referrer Abuse-cases

The same referrer information that can be used for legitimate reasons, can be abused by web-site operators to assault a user's privacy and in certain cases even perform user impersonation attacks.

- **Tracking visitors**: Traditionally, users associate tracking with tracking cookies. A web-site that wishes to track its users between page loads, or collaborating sites that wish to track users as they transition from one to the other can do so through the referrer header even if users delete their cookies on a regular basis.
- **Session Hijacking**: As described in Section 2.1, the referrer header contains not only the domain of the page where each request originated but the full URI of that page. That becomes problematic when web-sites use GET parameters to store sensitive data, as is the case in sites that add session information to URIs instead of cookies. In this case, the HTTP referrer will contain the session identifier of the user on the originating site, allowing a malicious operator of the target web-site to impersonate the user on the originating site [19].
- **Sensitive-page discovery**: It is common for web-developers to hide the existence of sensitive files and scripts in directories that are accessible from the Web but are not linked to by any visible page of the web-site. These files or directories sometimes have obfuscated names to stop attackers from guessing them. Penetration-testing tools such as *DirBuster*[2] that attempt to guess valid directories using dictionary and brute-force attacks, attest towards this practice. In such cases, if one of the sensitive pages contains an external link or external resource, the exact location of that page can be leaked to the remote web-server through the referrer header.

3 Referrer-Anonymizing Services

In Section 2 we described the possible uses and abuses of the HTTP referrer. Given the wealth of information that the referrer header provides, one can think of many scenarios where the user doesn't want to release this header to remote web-servers.

Today, users can achieve this either by configuring their browser not to send the referrer header, or through the use of Referrer-Anonymizing Services when clicking on links from sensitive web-pages. While the former approach is available on many modern browsers, it works as an all-or-nothing setting in the sense that the user cannot selectively allow the transmission of the referrer header. This can be problematic when a user navigates to web-applications that use the referrer header as a CSRF countermeasure. In such cases, the user wouldn't be able to use the web-application, unless they re-enable the transmission of the referrer

[2] `http://sourceforge.net/projects/dirbuster/`

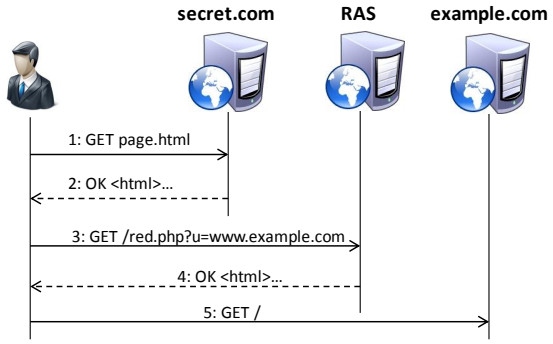

Fig. 1. HTTP messages involved in the use of a Referrer-Anonymizing Service

header. An additional problem is for web-site owners that wish to link to third-party sites but not at the expense of uncovering their identity. A controversial but popular example are 'warez forums'[3], where the descriptions of the pirated software or multimedia are usually given by linking back to the legitimate web-sites. In these cases, the web-site operators cannot rely on privacy-aware users, but must use a solution that will seamlessly work for all. This can be achieved through the use of Referrer-Anonymizing Services.

A Referrer-Anonymizing Service (RAS) is a web-service that is responsible for anonymizing the referrer header of a user before that user reaches a remote web-server. Note that, for security reasons, a web-site is not allowed to arbitrarily change a user's referrer header. The referrer header is created and emitted by the browser and thus the only way to anonymize this header is for the RAS to place itself between the site that links to an external resource, and that resource. By doing so, the RAS appears in the referrer header of the user's browser, instead of the original web-site, thus effectively anonymizing the original web-site. This technique is conceptually similar to the anonymizing techniques applied by Crowds [23] and TOR [11] where instead of the user's IP address, the link-embedding web-site is hidden.

Figure 1 shows the series of steps involved when using a RAS. In steps 1 and 2, a user requests and receives a page from `secret.com`. This page has a link that, if clicked, will eventually lead to `example.com`. However, since `secret.com` wishes to remain anonymous, it uses a RAS instead of linking directly to `example.com`. In step 3, the user clicked on the link expecting that it leads to `example.com`. However, the link creates a `GET` request towards a RAS with `example.com` as its argument. In response, the RAS generates a page (step 4) that will automatically redirect the user to `example.com` either directly, or after a timeout. In both cases, as far as the user's browser is concerned, the final request towards `example.com` originated not from `secret.com` but from the web-site of the RAS. Thus, the request depicted in Step 5, will have the redirect-causing web-page of RAS as its referrer, effectively hiding the original source of the link. Note that in the

[3] 'warez' is slang for pirated software

aforementioned process, `secret.com` will still reveal its presence to an external entity, but it chooses to reveal itself to the RAS instead of `example.com`.

The RAS can redirect the user's browser to `example.com` using one of the following ways:

- **HTTP MOVE messages**: When a web-server receives a request for a resource, it can emit a 301/302 `HTTP MOVE` message, that informs the user's browser of the 'move' of the resource, and provides it with the new location. Upon the receipt of such a message, a browser automatically initiates a new request towards the instructed location, thus completing the redirect.
- **HTML Meta-Refresh tag**: One tag of the HTML specification allows the web-developer to 'refresh' a page after a configurable number of seconds. The refresh can load the same page, or a new one. For example, `<meta http-equiv="refresh" content="5;url=http://www.example.com">` instructs the user's browser to replace the current page with the main page of `example.com` upon the expiration of 5 seconds.
- **JavaScript**: The same effect can be achieved using JavaScript, by setting the value of the `window.location` property to the desired site.

4 Taxonomy of RASs

In this section we analyze and compare common features of real-world Referrer-Anonymizing Services. We obtained a list of 30 functional RASs (listed in the appendix) by using a well-known search engine and searching for phrases related to their services, such as 'referrer anonymization' and 'hiding referrer'. The popularity of some of these services is evidenced by their high ranking in the Alexa top sites list. For instance, the most popular RAS, `anonym.to`, currently ranks higher than well-known sites such as `blackberry.com` and `barnesandnoble.com`. We summarize the discovered features of the studied RASs in Table 1.

4.1 Redirection mechanism

By manually visiting and recording the redirection mechanisms of the 30 RASs, we found out that 73% of them were redirecting their users using the meta-refresh mechanism, 20% using JavaScript and 7% using a combination of both 302 and meta-tags. The use of the HTML meta-refresh is the most common mechanism because it doesn't require JavaScript to execute and because it allows the RAS to delay the redirect in order to show advertising banners to the visiting users. The sites that used JavaScript to redirect a visitor, used it together with a timeout function to emulate the effect of the meta-refresh mechanism.

The HTTP MOVE messages were the least used among the services for the two reasons. Firstly, redirects occurring through a 301/302 HTTP message retain the original referrer header and are thus not suited for use from RASs. The services that did utilize them, always combined them with a meta-header, where the 302 message would redirect the user to another page on the RAS's web-site

Common Feature	Percentage of RASs
Redirection:	
HTML meta-refresh	73%
JavaScript	20%
HTTP MOVE+ meta-refresh	7%
Ads	36.66%
Mass Anonymization	50%

Table 1. Common features of Referrer-Anonymizing Services

which would then use a HTML meta-refresh tag. Secondly, even if the browser would clear out the referrer header, the HTTP MOVE mechanism doesn't allow for a delayed redirect, thus the services cannot use it to show advertisements.

An interesting observation is the diverse redirection behavior that different browsers display. When using Mozilla Firefox (version 9) and Google Chrome (version 16), a redirect implemented through JavaScript retains the referrer that caused the redirect. That is not a problem for RASs since the page that causes the redirect is not the original page that wishes to remain hidden, but a page of the RAS (step 4 in Figure 1). On the other hand, the same redirection mechanism in Microsoft's Internet Explorer 8, clears out the referrer. Contrastingly, Firefox and Internet Explorer clear out the referrer header in case of a HTML meta-refresh but Chrome still retains the redirect-causing referrer. From a point of user privacy, the complete clearing of the referrer is the best option for the user since the web-server cannot distinguish between users coming from web-sites that protect themselves and users who typed in the URIs or clicked on their browser bookmarks. However, the same mechanism that protects a user's privacy may negatively affect a user's security, as later explained in Section 8.

4.2 Delay and Advertising

Since all RASs that we encountered were providing their redirection services for free, there is a high probability that they attempt to capitalize on the number of incoming users through the use of advertising. From our set of 30 services, 11 (36.66%) were displaying advertising banners to the users waiting to be redirected to the destination web-site. From these services, 10 of them were constructing advertisements on the fly (through the use of client-side scripting and a chain of redirections) and only one had the same banners, statically embedded in its web-site. We also noticed that the sites that included advertising had, on average, a higher delay than the non-advertising web-sites which sometimes didn't delay the user at all. More specifically, the RASs with advertisements were redirecting the user after an average delay of 11.8 seconds whereas the non-advertising RASs were redirecting the user after an average delay of 2 seconds.

An interesting observation for the RASs that create dynamic advertisements is that all the requests towards the advertising agencies contain a referrer header which is the URI of the RAS page where the user is waiting to be redirected to the destination site. Since all RASs work by receiving the destination URI

(`example.com` in Figure 1) as a GET parameter, the various advertising agencies get access, not only to the IP addresses of the RAS visitors, but also to their eventual destination. By combining this knowledge with other data, they may be able to associate users with sites, even if the destination web-site doesn't collaborate with a specific advertising agency. Thus, in one third of the cases, the privacy of individual users is sacrificed for the anonymity of the linking site.

4.3 Mass Anonymization

Most RASs have a simple API for use of their services. All a RAS needs, is a remote URI to which it will redirect users while clearing, or substituting, their referrer header. For this reason, all RASs work in a stateless fashion. Unlike URL shortening services, where a user needs to first visit the service and generate a new short URL for their long URL, a user can utilize a RAS without first visiting the RAS's web-site. In our example in Figure 1, the administrator of `secret.com` can create an anonymized-referrer link to `example.com` simply by making a GET request with `example.com` in the `u` parameter.

This stateless nature of RASs allows for mass-anonymization of links without the hassle of registering each and every link to a remote service. From the 30 RASs that we analyzed, 50% were providing a mass-anonymization option through the use of an anonymizing script. This script, which is supposed to be included by remote web-sites, iterates over all `<a>` elements of the current HTML page and converts all links to RAS-links. Additionally, the scripts usually provide a white-list option where domains that do not need be anonymized (such as the links to local pages within a web-site) can be listed and excluded from the anonymization process. While we didn't encounter a misuse, a site including a remote anonymizing script is implicitly trusting the RAS providing it to not include malicious JavaScript along with the anonymizing functionality.

4.4 Background Activity

Apart from advertisements, RASs can use the browsers and IP addresses of visiting users to conduct arbitrary requests before redirecting the users towards their final destination. This activity can range all the way from harmless but unwanted to malicious. In our analysis of 30 RASs, we found that 2 services were performing unexpected actions that were hidden from the user.

The first service had embedded an invisible iframe that performed a search request with the keyword 'myautopass' using Google Search. While the RAS cannot steal any private data from that iframe (since the Same-Origin Policy disallows such accesses), the requests were made by the user's browser and user's IP address. As far as Google Search is concerned, tens of thousands of people [4] search for that word on a daily basis, an action which most likely affects the ranking and trend of that keyword, even if the shown links are never clicked.

[4] We obtained the RAS's estimated number of daily visitors using `quantcast.com`

The second service, instead of redirecting the user to the desired web-site, created a frameset with two frames. In the first frame, which spanned the entire page, it loaded the requested site and on the second frame it loaded a local page of that RAS. In this scenario, while the user gets access to the remote page, they never actually leave the site of the RAS. This 'sticky' behavior is common in anonymizing web-proxies which request a page from a remote server on the user's behalf and then present the result to the user. To the remote web-server, the request appears as coming from the anonymizing proxy's IP address, thus hiding the user's IP address. Note however that in the case of RASs, this 'sticky' behavior adds no more privacy to the visiting user, since the requests are all made from the client-side and thus using the user's browser and IP address.

By analyzing the contents of the second frame, we observed that through a series of redirects, the site was opening the registration page of a file-hosting web-site, with a specific affiliate identifier. It is unclear how the operators of that RAS were expecting to fill in the registration page in an invisible frame but this method hints towards the RAS's attempt to capitalize on visiting users for monetary gains in affiliate programs.

5 Information Leakage

Whenever a user utilizes an online service, there is always the possibility that the service will retain information from the user's activity and use that information in the future, possibly for financial gains. In Section 4.2 we showed that 36.66% of all tested Referrer-Anonymizing Services used ads as a way of getting monetary compensation for their free services. In almost all cases, the ads were created dynamically, initiating a GET request for a script or an image from the RAS to the advertising company. Through the referrer header, these requests reveal to the advertising agencies which page the user is on (the RAS waiting page) and the page that the user intends to go to (the destination GET argument given to the RAS). This is problematic for two reasons: first, if the destination URI contains a secret parameter tied to a specific resource (e.g. a session identifier, a file identifier for file-hosting services or a document identifier for online collaboration platforms) this identifier will be disclosed to an untrusted third party (the advertising agency). The second reason is that advertising agencies gain more information about users and the sites they visit even if the destination sites do not collaborate directly with them.

Another instance of the same problem is encountered between the source site and the RAS. The administrator of a RAS is able to view the referrer headers of traffic towards their service and can thus discover the original pages that relay visitors through them (e.g `secret.com` in Figure 1). If the source site hosted the link to the RAS on a page with sensitive data in its URL (both path and GET parameters) – e.g. `secret.com/admin.php?pass=s3cr3t` – this will be available for inspection to the utilized RAS.

In order to measure whether the various advertising agencies of RASs make use of users' referrer headers and whether the administrators of RASs access sensitive source pages, we conducted the following experiments.

5.1 Experimental Setup

We first created and registered `fileleaks.co.cc`, a web-site supposedly providing leaked sensitive documents and then developed two crawlers that visited all the RASs daily and requested a redirection towards URIs within our site. e.g. `http://anonym.to?http://fileleaks.co.cc/index.php?filename=[POPULAR_TOPIC]&dlid=[PER_RAS_UNIQUE_ID]`. The `dlid` contained a random identifier that was unique for all tested RASs and allowed us to accurately detect which RASs leaked our destination site.

The first crawler was simply requesting the URI in a `wget`-like way and proceeding to the next RAS. In this case, our destination URI could be leaked only through the web-server logs of the target RAS since no scripts or images were rendered. The second crawler was actually an instrumented instance of Firefox that automatically visited each site and waited for 10 seconds before moving on to the next target. The key difference between the two crawlers is that the latter one was a functional browser which executed all needed image and JavaScript requests to fully render each page. These two crawlers allowed us to roughly distinguish between URIs leaked by the web-administrator of a RAS and URIs leaked through the referrer header sent to advertising agencies.

To detect RAS administrators looking for sensitive pages in the referrer headers of their logs, we added a fake password-protected administrative panel to our site and programmed an additional `wget`-like crawler which constantly visited all RASs, pretending that the request for anonymization was originating at `http://fileleaks.co.cc/admin/index.php?password=[SECRET_PASS]&pid=[PER_RAS_UNIQUE_ID]`. The fake administrative script was logging all accesses and the `pid` GET parameter was used to distinguish leakage between the tested RASs as in our first set of crawlers.

5.2 Results

Leakage of destination site: In a 30-day period our monitors on `fileleaks.co.cc` recorded a total of 250 file requests using unique URIs that were made available to the Referrer-Anonymization Services as destination sites. By decoding each URI we identified that the URIs were leaked by 3 different RASs. Interestingly, all the recorded URIs were communicated to the services through our instrumented Firefox crawler and not through the `wget`-like crawler, implying that the URIs were most likely leaked by subsequent JavaScript and image requests of each RAS-waiting page. For privacy and ethical reasons, we do not identify the services by name and we refer to them as RAS1, RAS2 and RAS3.

The unique URI of RAS1 was found in 1.2% of the requests. It appears that the service leaked the URIs directly to a specific search engine, which at a later time requested the files with the exact parameters originally provided to

RAS1. RAS2 and RAS3 were both leaking the requests towards services running on Amazon's Elastic Compute Cloud. The requests leaked by RAS2 (88.8% of the total requests) were revealing, through their user-agent, that they were crawlers working on behalf of a specific advertising company which specializes in identifying which pages would prove to be the best ones for ad placement for any given product or service. The last set of requests (10% of the total) with leaked identifiers from RAS3 were also originating from hosts on Amazon's Cloud but their user-agent didn't provide any identifying details.

In total, our experiment showed that 10% of the tested RASs were, knowingly or not, leaking the referrer-header of their users to third-party advertising agencies who were recording them and using them at a later time. Given the risks associated with the leakage of a visiting user's IP address and destination site, we believe this to be a significant privacy risk.

Leakage of originating site: In the same period, our administrative-panel monitor recorded three visits from the same visitor. In the third visit, the user provided the exact `password` and `pid` combination that our third crawler was providing one of the 30 tested RASs through its referrer header. It is important to realize that given the nature of our crawler, the referrer header containing the `fileleaks.co.cc` administrative panel URI (and password as GET parameter) could only be retrieved from the RAS web-server's logs since the pages were always retrieved but never rendered in a real browser. Thus, no advertising agencies, or legitimate Web traffic scripts could ever access our referrer header. This shows that the administrator of one of the thirty RASs was actively searching the logs of the RAS in an effort to identify 'interesting' source sites and then manually inspect them.

6 User Categorization

In the previous section, we explored the various features of RASs and recorded which traits are prevalent and for what reasons. While this gives us an insight of the motives behind these services, it doesn't provide any specific details on the users who use these services or the actual reasons justifying their usage. In order to investigate the user-part of the RAS ecosystem, we created and advertised our own Referrer-Anonymizing Service, which we made available at `www.cleanref.us`. In a period of 25 weeks, our service received a total of 2,223 requests for referrer-anonymization. Figure 2 shows that the weekly usage of our service varied significantly. In the next sections we describe a subset of these requests according to their purpose.

6.1 Ethical considerations

The data that was collected for this experiment are the following: For each request we recorded i) its timestamp ii) the IP address of the host performing the request, iii) its GET parameters and iv) the referring host. These were collected

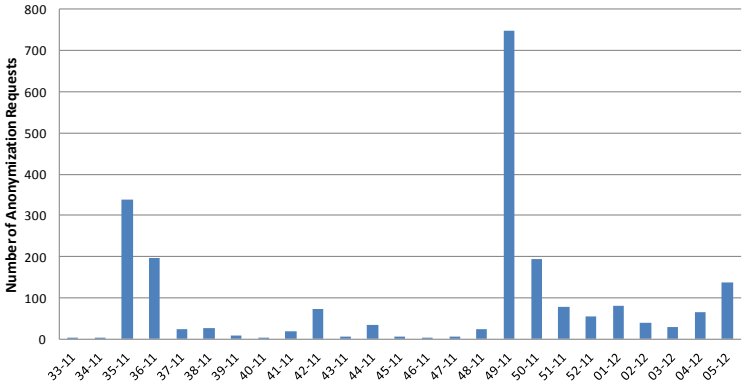

Fig. 2. Weekly number of anonymization requests that our RAS received during our study

in a single text file on the web server, in a password-protected directory that only the authors of this paper had access to.

The data collected is a subset of the data collected by every web server on the web in standard server logs. Many web developers even share this collected information with third parties, such as Google Analytics, for the purpose of gathering usage statistics. The reason for deploying our own RAS was to identify potential abuse of such services. Since reporting to users up front that they would be part of an experiment would defeat the purpose of the experiment, our RAS did not explicitly warn users of the data collection. Because of the nature of the data collected, and the fact that these data are collected by every web server, we believe this lack of user warning to be acceptable.

We are not planning on releasing the data to the general public, and we will delete all data after publication of the paper.

6.2 Hiding advertising infrastructures

The greatest part of the first peek of Figure 2 corresponds to 317 requests conducted by 127 unique IP addresses. By analyzing our data, we discovered that for two days in the 35th week of 2011, our service was part of two advertising campaigns and the requests were from users who were the targets of these campaigns. While a significant number of countries was involved, the campaigns seem to have been targeting users mostly in Vietnam and the US, since these countries received 31.49% and 16.53% of the total traffic, respectively. It is unclear whether the advertising campaign was conducted through spam emails or through browser pop-up windows, however the fact that the advertiser used a Referrer-Anonymizing Service shows that they wished to conceal the exact method of driving traffic by hiding the referrer header from the final web-servers.

In our referrer logs for those days, we found 3 types of links. One type of link, was used to drive traffic directly from our service towards the destination

web-site. The second type of link, was a chain of RASs all connected together in a way that allowed each RAS to redirect the user to next one, until the user is redirected to their final destination. For example, when the link:

```
http://cleanref.us/?u=http://www.refblock.com?http://cloakedlink.com
/zqkzqrfvgs
```

is clicked (or opened in a pop-up) by a user, her browser will request a page from `cleanref.us` which will redirect her to `refblock.us` which in turn will redirect the user to `cloackedlink.com`. `cloackedlink.com` is the end of the chain and when resolved, it will redirect the user to the actual site. The combination of multiple RASs allows the advertiser to hide its presence behind multiple layers of indirection. For instance, using this method, an inspection of the referrer does not reveal whether `cleanref.us` was the first part of a chain or whether another RAS redirected the user to our service using a redirection method that completely cleaned the referrer header. The last type of link, pointed to a service that received an obfuscated string as its only parameter:

```
http://linkfoobar.net/track-link.php?id=aHR0cDovL3d3dy5jbGVhbnJlZi51c
y8/dT1odHRw0i8vd3d3LnJlZmJsb2NrLmNvbT9odHRw0i8vY2xvYWtlZGxpbmsuY29tL3
pxa3pxcmZ2Z3M=
```

The `id` argument passed to the anonymized `linkfoobar.net` web-site is a Base64-encoded string that, when decoded, makes a chain of RASs similar to our previous example. As the name suggests, this is most likely the first part of the chain where the advertiser first tracks that someone clicked on a tracked link and then proceeds to redirect the user to the final destination through a chain of RASs.

By combining the knowledge of all three categories of referrer URIs found in our logs, it is evident that the advertiser mixes the order of RASs in their chains in order to reveal only part of their traffic and infrastructure to each RAS. Thus in some cases, our service was the last in the chain of RASs, sometimes in the middle and occasionally the first service that began the chain of RASs after the user's click was registered by `linkfoobar.net`.

6.3 Remote image linking

Throughout our experiment we noticed several groups of requests (e.g. weeks 42-11, 04-12 and 05-12 in Figure 2) towards image files, some of them located on popular image hosting sites. By sampling some of the destination URLs, we noticed that the requested images were almost always of an adult nature. For the sampled requests, we also reviewed the referrer header when it was available. The observed linking sites fell into two categories. In the first category, the sites linking to adult images through our RAS were forums where the users could write new posts and include links to images. In this case, the RAS was added to hide the linking site from the image hosting site, since the uploaded images

didn't conform with the rules of the latter. Some of the requests were using more than one RAS chained together as shown in Section 6.2.

In the second case, the linking sites were personal sites that were requesting images either from image hosting sites or directly from adult-content sites forming a client-side image collage. As in the former category, the owners of such pages were hiding the exact location of their personal pages from the sites hosting the linked images.

6.4 Web-mashups

A web-mashup is a combination of data and functionality from more than one remote service, which has more value than any one of the remote services by itself. The highest peek in Figure 2 stems from the adoption of our service from a book price-comparison web-application. The destinations of these requests are popular online bookstores and other price-comparison sites in Europe and the US. Each request contained the ISBN number of a different book. In a period of 9 weeks, the web-application initiated a total of 1,273 requests for anonymization of which 1,198 (94.10%) formed a unique combination of ISBN number and third-party service. Given the vast majority of unique queries for books, we believe that the requests happen once and their results are cached in the web-application. The usage of a RAS in between the book price-comparison web-application and the other online bookstores, allows the former to retrieve information from the latter without revealing its location or purpose.

7 Tracking of Anonymizers

In the previous sections, we analyzed the existing Referrer-Anonymizing Services, we listed some ways that one can legitimately or illegitimately use them and provided empirical data on the types of users that they attract. The final part of the RAS ecosystem are the destination sites (`example.com` in Figure 1). It is interesting to know, whether popular web-sites are aware of the existence of RASs and if they are, how do they react towards traffic relayed through them.

In order to identify differences in served content, we conducted an automated experiment involving the top 1,000 sites of the Internet according to Alexa. The first time we visited each Alexa link, we provided the URL of a popular search engine as our referrer header, simulating a user who followed a search result from that search engine. We then repeated the same request 30 times, each time providing as a referrer, the URL of one of our 30 RASs.

Given the dynamic nature of the Web, simply comparing the pages of different visits or their hashes is insufficient to differentiate between changes that were due to the web-site's reaction to a certain referrer header and usual expected changes, such as different timestamps or randomly generated data embedded in the source-code of the page. In order to overcome this obstacle we used a combination of two methods. The first method was to apply Arc90's *Readability* algorithm [3] which attempts to separate and show the most important content

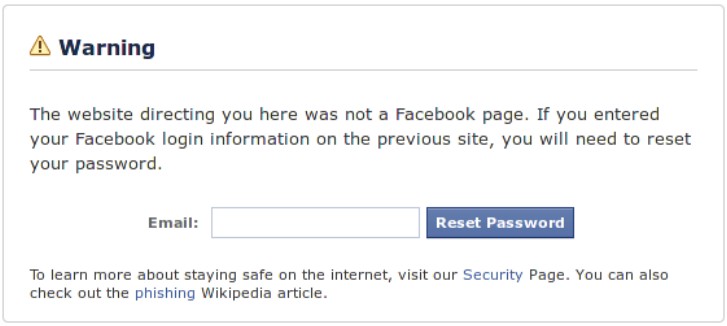

Fig. 3. Facebook's main page when visited through a specific RAS

on a page while hiding the less important. In the second method, we recorded the number and type of HTML input elements present on the page. The rationale behind the second method was that, even if a page legitimately changes between successive visits, the number of visible and invisible input elements should not change. If any one of the two methods provided different results between the first search-engine-referred visit of the site and any of the the RAS-referred ones, the site and its HTML code was recorded and the results were manually verified.

From a total of 1,000 web-sites we discovered that three of them were using the referrer header to provide radically different content. The first, `facebook.com`, was serving a very different page when our requests claimed to come from one of the 30 studied RASs. By manually checking the resulting page, we realized that instead of Facebook's usual login-screen, we received a page that alerted us that we had most likely been a victim of a phishing attack, and was inviting us to start the procedure of resetting our password (Figure 3). Interestingly, `facebook.com` reacted this way only when the referrer header was coming from that specific RAS and provided the normal content for the remaining 29 RASs. This could mean that Facebook was reacting to a real phishing attack where the attackers were trying to hide the location of the phishing page by redirecting their victims back to `facebook.com` through that specific RAS after the user credentials had been stolen.

The second web-site was a photo gallery where users can upload pictures and retrieve links that they can later use in various web-sites such as forums and blogs. When the site was visited through one of the 30 RASs, instead of giving us access to its main page, it provided us with a GIF image that stated that the site had blocked access to the linking page. This picture will be provided to any image requests that pass through that RAS. This verifies the behavior that we discovered in our own RAS, where visitors were linking to adult content uploaded to generic image galleries and hiding the linking site through the anonymization of their referrer header. The third site was a general information portal which was consistently providing a 403 HTTP 'forbidden' error when visited through a specific RAS, the same RAS blacklisted by the image gallery site.

The above experiment shows that even though only a small fraction of the tested web-sites reacted visibly to specific referrers, their behavior was always 'negative' when the referrer appeared to be a RAS. This attests towards the argument that RASs are associated more with illegal activity and less with a legitimate user's privacy concerns.

8 Related Work

Although the practice of cleaning the 'Referer' header through a RAS is common knowledge, we are unaware of any research into the operation or usage of these services, with regard to online privacy and anonymity. The related work that we are aware of, falls in the following 4 categories:

Online privacy and anonymity Online privacy and anonymity are important for numerous reasons. The Internet was not built to provide anonymous communication mechanisms, which lead to the creation of various projects that provide forms of anonymous networking. The Onion Router (Tor) [25] project and the Invisible Internet Project (I2P) [16] are the most famous of these networks.

Experiments It is easy to state that a security problem is real and dangerous. Providing evidence backing up this claim is often difficult since it involves covertly tracking the behavior of attackers and victims in an ethical way.

In our experiments in Section 5, we used enticing links to lure visitors to our own *fileleaks.co.cc* in a honeypot-like way. Honeypots [22] have been traditionally used to study attacking techniques and post-exploitation trends. Yuil et al. [29] introduce Honeyfiles as an intrusion detection tool to identify attackers. Honeyfiles are bait files that are stored on, and monitored by, a server. These files are intended to be opened by attackers and when they do so, the server emits an alert. Similarly, Bowen et al. [7] use files with 'stealthy beacons' to identify an insider thread. We have used these techniques in the past to detect whether attackers are violating the assumed privacy in file-hosting services [20].

Referrer abuse Referrer headers were designed to identify which URL a visitor is coming from. This information is of great benefit to content-providers because it can provide some insight in the browsing habits of visitors, as discussed in Section 2.2. The referrer data however, is optional and can be spoofed (e.g. RefSpoof [12]), prompting the inception of *referrer spam* [27]: sending the URL for a third-party web-site in the referrer header so it will show up in the logs of a visited web-site.

Because of their use to track visitor movements, everyone naturally expects referrer headers to contain URLs. This expectation can result in the development of a web-application which displays the contents of the referrer header, without sufficient sanitization. In such an application, the referrer header can be abused to carry an XSS attack as documented in [14].

Many Cross-site Request Forgery (CSRF) countermeasures depend on the referrer header to determine whether a request was sent from a trusted location.

The authors of some of these CSRF countermeasures, aware of visitors that disable the transmission of referrer information, will implement lenient referrer validation [5], which will allow requests without referrer header in order not to break the web-application that is being protected. This deliberate loophole allows an attacker to launch a CSRF attack by relaying a request from an untrusted location through a RAS, which will remove the referrer information. Because the request in this attack has no referrer information, it is allowed by the CSRF countermeasure and the attack can succeed.

Solutions dealing with referrers There are only two parties that can benefit from non-disclosure of referrer information: the visiting browser and the author of the web-site on which a link is hosted. The referrer-leakage problem can thus be solved by either party.

Referrer-Anonymizing Services attempt to solve privacy and anonymity issues that arise because a visitor's browser is leaking information through the referrer header by design. The author of a web-site linking to an external web-page used to not have any other means to prevent the referrer header from exposing their web-site. One way the author of a web-site could prevent the referrer header from exposing their web-site, was to host their web-site using HTTPS. The HTTP protocol specification [24] advises that referrer information should not be sent when navigating from an HTTPS web-site to an HTTP site. However, browsers are free to preserve the referrer if the destination is also an HTTPS web-site, even if the destination site is situated on a different domain.

Recognizing the need for a better solution, the WHATWG has included the 'noreferrer' link type [26] in the HTML5 specification. By annotating certain HTML elements with this link type, a web-page author will prevent referrer information from leaking when clicking the annotated link. RASs protect the web-site's anonymity as much as they protect a visitor's privacy. Therefore it makes sense for an Internet user to disable referrer information to safeguard that privacy at the source. Many modern web-browsers provide means to disable referrer header transmission [18, 6, 21]. For other browsers, the referrer can be filtered out using a client-side proxy like e.g. Privoxy [10]. Due to the privacy problems associated with the referrer header, the 'Origin' header [4] has been proposed because it only leaks the origin (scheme, hostname and port number) of a URL to a remote web-site instead of the full URL.

9 Conclusion

In this paper, we explored the ecosystem of Referrer-Anonymizing Services and classified their functionality, their user-base and their abuse. We showed that in several cases, RASs were taking advantage of their position and leaked private user information to advertising companies. Conversely, we discovered that users were occasionaly using RASs to hide illegal or unethical activity and we revealed that some popular Internet sites do not respond well to RAS-relayed traffic. Overall we showed that, while protecting a user's privacy through the

anonymization of the referrer header is desirable, not all RASs are equally noble and thus care should be taken when choosing one. At the same time, browser developers have the responsibility to facilitate a complete migration away from such services through the support of privacy-preserving HTML5 tags.

Acknowledgments: This research is partially funded by the IBBT, the Research Fund K.U.Leuven, the B-CCENTRE, the EU-funded FP7 project NESSoS and the IWT project SPION.

References

1. Panopticlick. `http://panopticlick.eff.org/`.
2. G. Aggrawal, E. Bursztein, C. Jackson, and D. Boneh. An analysis of private browsing modes in modern browsers. In *Proc. of 19th Usenix Security Symposium*, 2010.
3. Readability — Arc90 Lab. `http://lab.arc90.com/2009/03/02/readability/`.
4. A. Barth. The Web Origin Concept. `http://tools.ietf.org/html/draft-abarth-origin-09`.
5. A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *To appear at the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, 2008.
6. P. Beverloo. List of Chromium Command Line Switches, –no-referrers. `http://peter.sh/experiments/chromium-command-line-switches/#no-referrers`.
7. B. Bowen, S. Hershkop, A. Keromytis, and S. Stolfo. Baiting inside attackers using decoy documents. *Security and Privacy in Communication Networks*, pages 51–70, 2009.
8. Z. Chu and H. Wang. An investigation of hotlinking and its countermeasures. *Computer Communications*, 34:577–590, April 2011.
9. R. Clayton, S. Murdoch, and R. Watson. Ignoring the great firewall of china. In G. Danezis and P. Golle, editors, *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 20–35. 2006.
10. P. Developers. Privoxy. `http://www.privoxy.org`.
11. R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.
12. ebricca. refspoof Firefox extension. `https://addons.mozilla.org/en-US/firefox/addon/refspoof/`.
13. P. Eckersley. How unique is your web browser? In M. Atallah and N. Hopper, editors, *Privacy Enhancing Technologies*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2010.
14. Recent referer xss vulnerabilities. `http://evuln.com/xss/referer.html`.
15. M. Fioravanti. Client fingerprinting via analysis of browser scripting environment. Technical report, 2010.
16. I2P Anonymous Network. `http://www.i2p2.de/`.
17. ietf-http-wg mailinglist. Re: Referer: (sic) from Phillip M. Hallam-Baker on 1995-03-09. `http://lists.w3.org/Archives/Public/ietf-http-wg-old/1995JanApr/0109.html`.
18. MozillaZine. Network.http.sendRefererHeader - MozillaZine Knowledge Base. `http://kb.mozillazine.org/Network.http.sendRefererHeader`.

19. L. Murphey. Secure session management: Preventing security voids in web applications, 2005.
20. N. Nikiforakis, M. Balduzzi, S. Van Acker, W. Joosen, and D. Balzarotti. Exposing the lack of privacy in file hosting services. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, LEET'11, Berkeley, CA, USA, 2011. USENIX Association.
21. Opera. Disabling referrer logging - Opera Knowledge Base. `http://www.opera.com/support/kb/view/93/`.
22. N. Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.
23. M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1:66–92, November 1998.
24. RFC 2616 - Hypertext Transfer Protocol.
25. Tor Project: Anonymity Online. `http://www.torproject.org`.
26. WHATWG. HTML - Living standard. `http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#link-type-noreferrer`.
27. Wikipedia. Referrer spam. `http://en.wikipedia.org/wiki/Referrer_spam`.
28. G. Wondracek, T. Holz, C. Platzer, E. Kirda, and C. Kruegel. Is the internet for porn? an insight into the online adult industry. In *Proceedings of the Ninth Workshop on the Economics of Information Security (WEIS)*, 2010.
29. J. Yuill, M. Zappe, D. Denning, and F. Feer. Honeyfiles: deceptive files for intrusion detection. *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*, (June):116–122, 2004.
30. W. Zeller and E. W. Felten. Cross-site request forgeries: Exploitation and prevention. Technical report, 2008.
31. Y. Zhou and D. Evans. Why Aren't HTTP-only Cookies More Widely Deployed? In *Proceedings of 4th Web 2.0 Security and Privacy Workshop (W2SP '10)*, 2010.

A Appendix

URI	Ranking	URI	Ranking	URI	Ranking
anonym.to	661	nolink.in	152,160	anonym2.com	846,517
hidemyass.com	904	cloakedlink.com	187,162	sock.im	933,195
referer.us	7,662	th3-0utl4ws.com	257,867	hidelinks.net	1,170,906
anonymizeit.com	10,212	savanttools.com	305,880	anon.projectarchive.net	2,591,907
lolinez.com	35,526	a.myurl.in	311,033	1-url.net	4,032,638
nullrefer.com	37,227	privatelink.de	338,512	crypo.net	5,009,009
linkblur.com	62,993	carcabot.com	347,260	anonym.ehmig.net	5,510,217
refblock.com	63,834	linkanonymizer.net	433,913	hidehelp.com	9,470,830
dereferer.ws	104,803	spoofurl.com	645,526	devgizmo.com	No Data
anonymous-link.net	118,261	refhide.com	679,101	theybetrollin.com	No Data

Table 2. Alexa Ranking of 30 tested RASs – gray color denotes RASs showing ads