Deactivation of Unwelcomed Deep Web Extraction Services through Random Injection

Varun Bhagwan, Tyrone Grandison
IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 USA
{vbhagwan, tyroneg}@us.ibm.com

Abstract

Websites serve content both through Web Services as well as through user-viewable webpages. While the consumers of web-services are typically 'machines', webpages are meant for human users. It is highly desirable (for reasons of security, revenue, ownership, availability etc.) for service providers that content that will undergo further processing be fetched in a prescribed fashion, preferably through a supplied Web Services. In fact, monetization of partnerships within a services ecosystem normally means that website data translate into valuable revenue. Unfortunately, it is quite commonplace for arbitrary developers to extract or leverage information from websites without asking for permission and or negotiating a revenue sharing agreement. This may translate to significant lost income for content providers. Even in cases where website owners are happy to share the data, they may want users to adopt dedicated Web Service APIs (and associated API-servers) rather than putting a load on their revenue-generating websites. In this paper, we introduce a mechanism that disables automated web scraping agents, thus forcing clients to conform to the provided Web Services.

1. Introduction

Many companies take the prudent approach when it comes to the development of systems that use online data from external parties, i.e. they undergo a formal documented legal process where consent is granted and a revenue package is agreed upon. However, these same companies tend to be frustrated when it comes to their own data being leveraged by other parties, who have less to lose and that do not take this prudent approach.

These companies normally employ web-scraping [1] to harvest their information. Formally defined, web-scraping [1] is the act of going through the content of a website for the purpose of extracting information from it. It is typically implemented by means of authoring an automated agent that makes appropriate HTTP requests to the website with the desired content, and 'scrapes' the said content from the result of the HTTP request (related issues have been dealt in detail in [2]). The scraping (or extraction or harvesting) is used to collect content such as user-data, image-links, user-comments, email addresses or any other data of potential value from the source website.

In the most malicious of cases, it can involve copying entire websites to direct traffic away from the source website and onto the (typically spam and or ad infested) malicious website. In this paper, we introduce a mechanism, Random Injection-based Deactivation (RID), to forcibly disallow automated web-scraping agents from harvesting or collecting data from a website.

2. Background

Figure 1 shows the HTML code that web scrapers would need to navigate in order to obtain image links for artists.

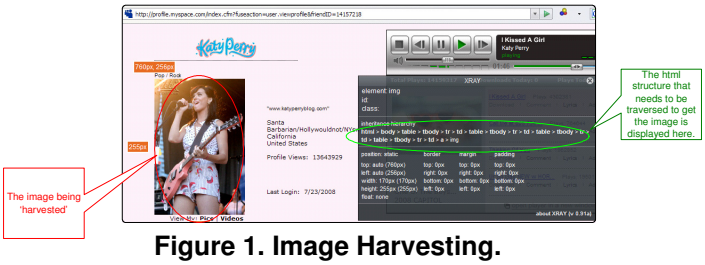

Figure 1. Image Harvesting.

Another example of the information that screen scrapers may be interested in is depicted in Figure 2, which shows the HTML hierarchy that the scrapers would navigate when extracting user data and comments.

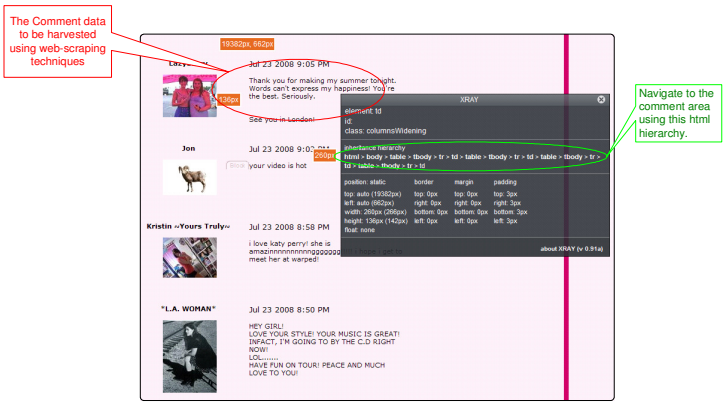

Figure 2. User Comment Harvesting

As stated previously, even if web-scraping is being done completely legitimately and for acceptable reasons, source websites may wish to divert traffic away from their main servers, and or to encourage such 'scrapers' to switch to using the provided Web Service APIs instead of scraping the (HTML) source code, whether for technical or

business/financial reasons. RID technology also enables this to happen.

Figure 3 shows the status quo in webpage provision today. Upon each data (HTTP) request, the source web server generates a dynamic page as a result of the HTTP request. In the end, the result, i.e. the HTML code, is presented to the end user and is easily harvestable by web scrapers.

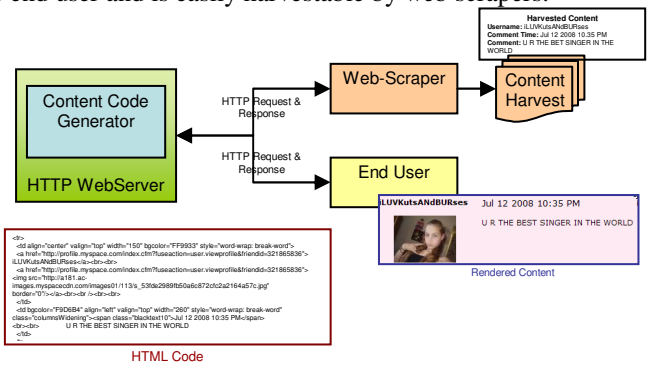

Figure 3. Typical Web Scraping

3. System

Figure 4 illustrates how RID technology may be included into the typical web server environment. In order to optimize the process of generating the dynamic HTTP request-result, the web-server 'pre-generates' the set of redundant code, and simply 'injects' randomly selected code into the base code at appropriate points (where the data needs to be hidden from web-scrapers).

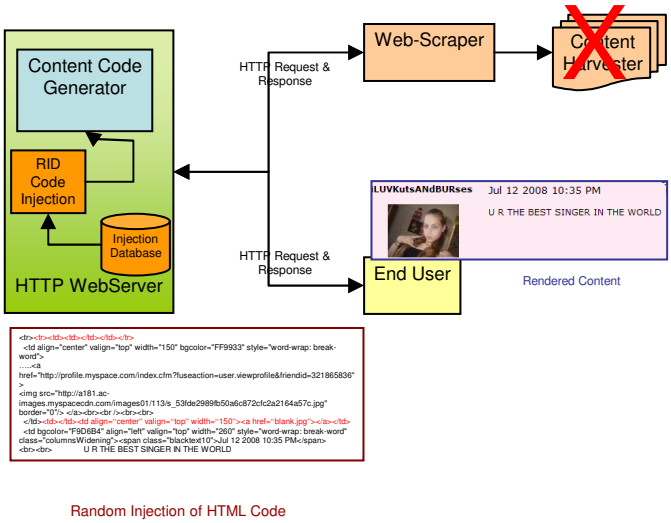

Figure 4. Random Injection based Deactivation (RID) of Web-Scrapers

As there is no change to the display on the screen, the end user experience remains the same, thus achieving the technology's objective of providing completely transparent and non-intrusive deactivation of web-scraping applications and services.

The following provides some insight into the inner operation of RID technology. Assuming that C is the set/list of content to be protected and that H is the HTML page normally rendered, the RID algorithm takes as input the content elements to be protected, C, i.e. content types (in the above scenario, this maps to image-files, user-comments etc.) and the (HTML) code that was rendered normally, H, and returns the new web scraper resistant code. As currently implemented, this technology is agnostic of the languages and vocabularies used for the rendered code as it focuses on the content elements to be protected. This design decision enables the system to work with new and emerging Web authoring technologies.

4. Related Work

Blocking IP addresses [3] stops both legitimate and illegitimate extraction from a (set of) IP address(es) and thus is not suitable when access is process-focused. Also, the random insertion of CAPTCHA [4] in scraping is common practice, which differs from our approach; in that our approach is completely transparent to legitimate users. Paid content sites (e.g. age appropriate entertainment) have a legacy of anti-crawling. These sites control access by means of user login. Other sites that make "small changes" or "hidden random strings" do not make their mechanisms "completely hidden" from end users.

5. Conclusion

The proposed system provides a way to disable web scraping by obfuscating the code rendered to the Web client, such that although the rendered webpage (as seen on the screen by the end-user) is unchanged the code behind the webpage is changed (dynamically) upon every fetch request. This code-poisoning technique ensures that no automated agent can reliably collect data from the website, thus rendering the extraction agent ineffective. In the end, the data consumer has no choice but to adopt the content provider's Web Services to gain access to the data.

References

[1] Web Scraping
http://en.wikipedia.org/wiki/Web_scraping

[2] Alfredo Alba, Varun Bhagwan, Tyrone Grandison: Accessing the deep web: when good ideas go bad. OOPSLA Companion 2008: 815-818

[3] IP Blocking,
http://en.wikipedia.org/wiki/Wikipedia:Blocking_IP_addresses

[4] Luis von Ahn, Manuel Blum, Nicholas J. Hopper and John Langford. "CAPTCHA: Using Hard AI Problems for Security". The proceedings of Eurocrypt 2003. Warsaw, Poland, May 4–8, 2003. LNCS Vol 2656/2003.