1-1-2012

CleanURL: A Privacy Aware Link Shortener

Daniel Kim
University of Pennsylvania, dki@seas.upenn.edu

Kevin Su
University of Pennsylvania, kevinsu@seas.upenn.edu

Andrew G. West
University of Pennsylvania, westand@cis.upenn.edu

Adam Aviv
University of Pennsylvania, aviv@seas.upenn.edu

CleanURL: A Privacy Aware Link Shortener

Abstract

When URLs containing application parameters are posted in public settings privacy can be compromised if the those arguments contain personal or tracking data. To this end we describe a privacy aware link shortening service that attempt to strip sensitive and non-essential parameters based on difference algorithms and human feedback. Our implementation, CleanURL, allows users to validate our automated logic and provides them with intuition about how these otherwise opaque arguments function. Finally, we apply CleanURL over a large Twitter URL corpus to measure the prevalence of such privacy leaks and further motivate our tool.

Disciplines

Computer Engineering

CleanURL: A Privacy Aware Link Shortener

Daniel Kim, Kevin Su, Andrew G. West, and Adam J. Aviv
Department of Computer and Information Science
University of Pennsylvania – Philadelphia, PA, USA
{*dki, kevinsu, westand, aviv*}*@seas.upenn.edu*

ABSTRACT

When URLs containing application parameters are posted in public settings privacy can be compromised if the those arguments contain personal or tracking data. To this end we describe a privacy aware link shortening service that attempts to strip sensitive and non-essential parameters based on difference algorithms and human feedback. Our implementation, CleanURL, allows users to validate our automated logic and provides them with intuition about how these otherwise opaque arguments function. Finally, we apply CleanURL over a large Twitter URL corpus to measure the prevalence of such privacy leaks and further motivate our tool.

1. INTRODUCTION

Uniform resource locators (URLs) often contain functionality that allows data to be passed to web applications. For example, consider the following URL:

```
http://www.example.com?key1=val1&key2=val2
```

There, the key-value pairs which proceed the question mark are collectively called the *query string*. Such parameters are common in practice for purposes of referrer tracking, user identification, and storage of session data.

Privacy concerns can arise when a user posts a URL containing a query string in a public forum (*e.g.,* a social network or micro-blogging platform) as parameters might contain/encode sensitive information. Moreover, since many posting environments are profile driven, a history of contributions could reveal considerable private user data.

When a parameter is interpreted by the server-side application it may or may not affect how the browser visually renders the page. Thus, privacy leaks occurring as the result of *non-rendered* parameters are avoidable given their purpose is orthogonal to that of link sharing, which is to share content. Certain users may be cognizant of this fact, but manual removal and inspection is an arduous task. More likely, most users are unaware of the threat. Thus, we aim to provide an intuitive and efficient means by which to visualize parameter output, educate about the potential dangers, and produce more secure links. This is realized as a privacy-aware link shortening service (CleanURL).

We begin by using visual and textual comparisons (*i.e., diff* algorithms) to determine whether or not a parameter's inclusion affects rendering. Dynamic webpage content (*e.g.,* advertisements) can blur such distinctions. Thus, we create an interface that allows end-users to visually confirm/correct our generated rankings regarding URL reduction (Sec. 2).

Over time, this feedback can be aggregated to refine our approach and increase usability by reducing service overhead. Moreover, we use our technique to conduct an unsupervised measurement study over a corpus of 1.6 million Twitter URLs. We found that half of the URLs have arguments, and of those, 63% contain non-rendered arguments; a cause for privacy concern (Sec. 3).

2. LINK SHORTENING SERVICE

In describing our link shortener, we begin by using diff algorithms to estimate parameter necessity (Sec. 2.1) before integrating these into a user-facing interface (Sec. 2.2).

2.1 Estimating Argument Necessity

Given two URLS (*i.e.,* one inclusive and another exclusive of some parameter) it is our goal to determine whether that parameter is a rendered one (*i.e.,* induces non-random content change). We consider two approaches:

- VISUAL DIFF: The two URLs are rendered as downscaled bitmaps with a standardized viewport and the Hamming distance between the images is calculated.
- HTML TAG DIFF: The HTML source of the two URLs is parsed to remove visible text content. Over the remaining content (*i.e.,* HTML tags) a standard textual diff is applied (based on longest common subsequence) and the size of the delta/patch is computed.

While visual difference is perhaps the most intuitive comparison method, one must also accommodate *dynamic content*. Even when an identical URL is fetched multiple times in succession the pages may render differently. Often, this comes in the form of different advertisement images whose prominent sizing can skew visual diff calculation.

The HTML tag diff provides an additional perspective. While visually different when rendered, advertisement placement code is generally quite static in nature (perhaps only changing the image path, if anything).

Both diff algorithms return real-valued measures that can be used jointly to determine the relevance of a URL argument. The values are aggregated and compared against a threshold to force a binary decision of whether the versions are sufficiently similar. This threshold should be selected such that it is tolerant of dynamic content (see Sec. 3.1).

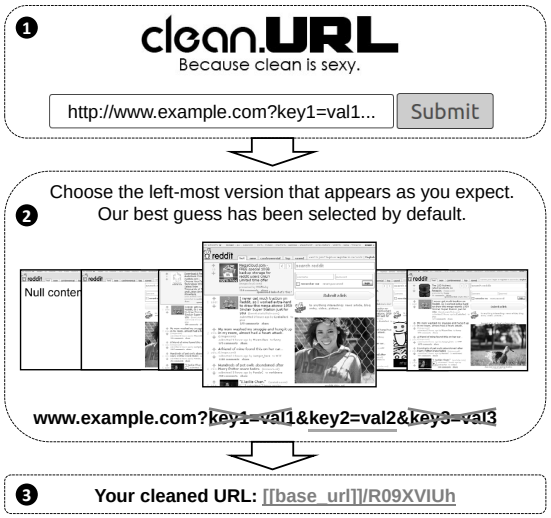

Figure 1: Simplified screenshots detailing end-user interaction with the CleanURL interface.

2.2 CleanURL User Interface

A typical session with the link shortener is visually depicted in Fig. 1. A user begins by entering a URL in a simple form field and pressing the "submit" button. This sets off the computational phase whereby all combinations of parameters from the query URL are enumerated. For each parameter combination the webpage source (*i.e.,* HTML) is downloaded, rendered as a small screenshot, and compared against the original URL via our diff functions. The diff output is used to sort combinations based on: (1) their ability to faithfully render the original URL, and (2) internal to that, the number of query string parameters.

This ordering is the basis by which screenshots are presented in a "shuffle" selector to the end-user (see Fig. 1). The "optimal" version is selected by default: the combination with the minimal number of parameters that still exceeds the similarity threshold. Our design goal was to visualize the impact of URL manipulation to achieve end-user awareness while still maintaining a clean, simple, and usable interface. The user is free to browse/shuffle through all presented combinations and a shortened URL (per standard hashing) is returned once a selection is made.

This act of human selection is essentially an evaluation of our automated technique. However, we could also imagine that humans might go a step further in choosing to eliminate parameters that *do* affect page rendering (*e.g.,* but are orthogonal to the page's main content). Such feedback is put to use to use shortly (Sec. 3).

CleanURL's implementation is coded in Django, a Python web application framework. Screenshot generation uses the QtWebKit engine supported by a VNC server.

3. UTILIZING FEEDBACK LOOPS

Feedback from the interface allows refinement of automated techniques to minimize users' time investment and heighten trust (Sec. 3.1). Confident in our system's abilities, the approach can be employed autonomously over a large corpus of URLs to estimate the prevalence and properties of privacy leaks "in the wild" (Sec. 3.2).

KEY NAME	PER%
`utm_source`	15.5%
`utm_medium`	14.6%
`utm_feed, feed, f`	12.4%
`ref, _r`	5.2%
`utm_campaign, campaign`	4.9%

Table 1: Most commonly stripped parameter keys. Percentage is out of total of all stripped instances.

N	0	1	2	3	...	8
CDF%	100%	52.5%	5.5%	1.9%	...	0.03%

Table 2: Percentage of URLs having $\geq N$ key-value pairs in the query string.

3.1 Usability Improvements

Human feedback establishes ground truth regarding "optimal" parameter selection. With these labels, the two diff values (Sec. 2.1) could be fed as features to a supervised machining technique to improve threshold selection. Given that CleanURL has not been deployed to the public we are yet to perform such optimization.

However, we believe the accuracy of CleanURL's automated logic is paramount in achieving service adoption given that human selection phase is the only overhead in using CleanURL relative to a traditional link shortener.

A popular system would also generate sufficient feedback to reason about parameter keys in a lexical fashion, *e.g.,* that "userid=" tags are typically stripped from submitted URLs. In this vein we have manually authored a static list of keys that tend to reveal private data (*e.g.,* `username`, `location`, *etc.*). If one of these keys cannot be safely stripped the user is warned, prompting manual inspection and possible abandonment (*i.e.,* the link is fundamentally insecure).

3.2 Motivational Measurement Study

We now use CleanURL's logic to demonstrate the real-world presence of privacy sensitive URLs. We manually set selection thresholds using a combination of our expertise, the diff score distribution, and a small number of inspections (*i.e.,* there is some margin of error).

We analyze a corpus consisting of 1.7 million URLs that appeared in Twitter posts. Tab. 2 shows the number of key-value pairs in query strings. We are fortunate that high pair quantities are rare as their combinatorial explosion would create interface latency. The figure also shows 52% of URLs (884,000 in total) have at least one parameter and these formed our working set moving forward.

Running these through the CleanURL system we found that 63% of URLs have at least one parameter stripped after inspection (*i.e.,* original URL vs. CleanURL's "optimal" prediction). Clearly, the presence of non-rendered parameters is pervasive. We also inspect the key names which were removed, with Tab. 1 showing the five most common (grouping those of similar name). Notable is the fact that all five are used for tracking purposes measuring referring websites, ad campaign effectiveness, traffic analytics, *etc.* Thus, not only can the parameters be safely removed, but it should be apparent their removal would positively impact the privacy of the CleanURL user. □