May, 2014

On the Privacy Concerns of URL Query Strings

Andrew G. West
Adam J. Aviv

On the Privacy Concerns of URL Query Strings

Andrew G. West
Verisign Labs
Reston, VA, USA
awest@verisign.com

Adam J. Aviv
U.S. Naval Academy
Annapolis, MD, USA
aviv@usna.edu

***Abstract*—URLs often utilize query strings (*i.e.,* key-value pairs appended to the URL path) as a means to pass session parameters and form data. Often times these arguments are not privacy sensitive but are necessary to render the web page. However, query strings may also contain tracking mechanisms, user names, email addresses, and other information that users may not wish to reveal. In isolation such URLs are not particularly problematic, but the growth of Web 2.0 platforms such as social networks and micro-blogging means URLs (often copy-pasted from web browsers) are increasingly being publicly broadcast.**

This position paper argues that the threat posed by such privacy disclosures is significant and prevalent. It demonstrates this by analyzing 892 million user-submitted URLs, many disseminated in (semi-) public forums. Within this corpus our case-study identifies troves of personal data including 1.7 million email addresses. In the most egregious examples the query string contains plaintext usernames and passwords for administrative and extremely sensitive accounts. With this as motivation the authors propose a privacy-aware service named "CleanURL". CleanURL's goal is to transform input addresses by stripping non-essential key-value pairs and/or notifying users when sensitive data is critical to proper page rendering. This logic is based on difference algorithms, mining of URL corpora, and human feedback loops. Though realized as a link shortener in its prototype implementation, CleanURL could be leveraged on any platform to scan URLs before they are published or retroactively sanitize existing links.

I. Introduction

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

```
http://www.ex.com/content.php?key1=val1&key2=val2
```

contains a set of key-value pairs (or application parameters) which collectively are called the *query string*. Query strings are common, with our dataset showing 56% of URLs have 1+ key-value pair(s). These are used for a variety of purposes when retrieving web content, but not all parameters may be necessary or desirable for the faithful retrieval of a web document. Such strings may contain tracking metrics or more sensitive information such as user IDs, location data, or passwords.

In isolation such URLs are of minimal concern, at most susceptible to shoulder surfing attacks. The problem is massively exacerbated when URLs are shared and published online. The URL and the sensitive data in the query string then becomes available to marketers, spammers harvesting contact information, and cyber-criminals with nefarious intentions. It comes as no surprise that a tremendous quantity of URLs end up on the public web, in no small part due to a Web 2.0 culture increasingly characterized by social networking and information sharing [1]. Moreover, since many posting environments are profile driven a history of contributions could reveal considerable private user data [2].

In this position paper we argue that these impacts on user privacy are significant and prevalent. Additionally, we feel social platforms have been insufficient in curbing such leaks, despite being intuitive locales for privacy preserving logic.

To the best of our knowledge the privacy ramifications of URLs and their query strings has not been previously analyzed (Sec. II). However, others have examined broader security considerations of URL transformation services and argument passing. Link shortening services have been an area of focus, as their obfuscation of URLs has enabled phishing and other abuses [3], [4], [5]. Others have produced specifications for cross-organizational parameter passing [6] and described the intentional manipulation of key-value pairs [7].

Given this lack of prior work we substantiate our position by undertaking a proof-of-concept measurement study over 892 million user-submitted URLs (Sec. III). These URLs contain 1.3 billion key-value pairs which were analyzed for sensitive data. We find over a quarter-billion plaintext pairs involved in referral tracking, with more than 10 million pairs possibly revealing some form of demographic, identity-based, or geographical information. In extreme examples, user and password authentication credentials were found in plaintext. Given non-standardized naming conventions, these quantities represent the lower bound on what is clearly an issue of significant severity and scale. For example, 2000+ unique key labels have email address values, with our analysis only able to consider the most popular such keys.

Fortunately, the social platforms via which URLs are often published and/or shortened offer an opportunity to identify privacy violating URLs. We propose a *privacy-aware URL transformation service* called "CleanURL" that provides platforms and their end-users with a mechanism to identify, remove, and assess privacy trade-offs for an input URL (Sec. IV). In its prototype form our service is presented as a stand-alone link shortener. The algorithmic detection of privacy-sensitive and/or unnecessary arguments presents research challenges. Towards these challenges we describe preliminary work on the use of: (1) Visual and document difference algorithms to measure effects on page rendering. (2) Extrapolations from manual analyses about the expected format of sensitive keys and values. (3) Feedback loops from the interface to understand privacy trade-offs and human factors.

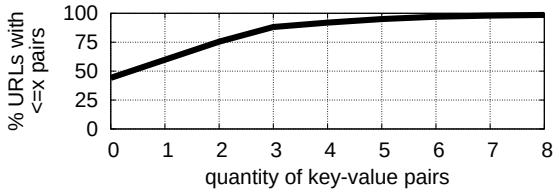

Fig. 1: CDF for quantity of key-value pairs in URLs

II. Related Work

Link shortening services are similar to our proposal in that they also transform input URLs. Such shorteners place a single redirection alias between a short link and its full representation. While convenient for presentation purposes and length constrained settings [1], shorteners do not sanitize links. Instead, these services provide one-hop of concealment/obfuscation for plaintext URLs. While this aids privacy by superficially keeping query strings from public view, it also prevents the raw URL from being interpreted by human users. This combined with the ease of link generation make shorteners a catalyst in a variety of attacks [8], including spam, phishing, and DNS fast-fluxing [3], [4], [5], [9], [10]. Tools like URL X-ray (`www.urlxray.com`) reveal the URL destination of these links without the consequences of clicking-through. Our proposed CleanURL service aims to strip sensitive key/values from URLs in an irreversible fashion; it is at the user's discretion whether these "clean" URLs are subsequently shortened.

To our knowledge no prior work directly addresses the privacy issues associated with URL query string parameters. That being said, early work did consider privacy when developing specifications for cross-organizational parameter sharing [6]. Moreover, [7] provides perspective on how these parameters are being utilized by server-side applications and might be manipulated. Finally, [2] discusses the privacy consequences of longitudinal tracking and data collection on the web, a phenomena enabled in part by query string use.

The CleanURL service needs to determine the affect of parameters on page rendering, even in the presence of dynamic content. Recent work in the Internet censorship domain describes the use of Merkle hash trees over page's DOM structure to detect such differences [11]. Vi-DIFF [12] takes a more visual approach to understanding the content and structural evolution of webpages. Our prototype currently prefers simpler heuristics, but either of these proposals could be viable improvements as our system matures.

III. Motivational Measurements

To demonstrate the privacy concerns of query strings and motivate the CleanURL service a large URL corpus is analyzed. That corpus is described (Sec. III-A) and examined for common query string use-cases (Sec. III-B). Then attention turns to key-value pairs with privacy implications (Sec. III-C) and their entropy (Sec. III-D).

A. Data Source & Summary

URLs for analysis were obtained from an industry partner which has access to a large quantity of URLs submitted directly by end-users. The nature of this partner service is such that it eases link tracking and handling, meaning the vast majority of submitted links are later found posted to Web 2.0 social and collaborative services. Thus, sensitive query string information is likely to find itself in the (semi-) public domain where it may be harvested by peers, marketers, or cyber-criminals. Our URL set consists of 892 million URLs, 490 million (54.9%) of which have 1+ key-value pairs (Tab. I). Some 5% of URLs have greater than 5 pairs and 23.4k URLs had more than 100 (Fig. 1). There are roughly 909k unique keys producing 1.3 billion total key-value pairs.

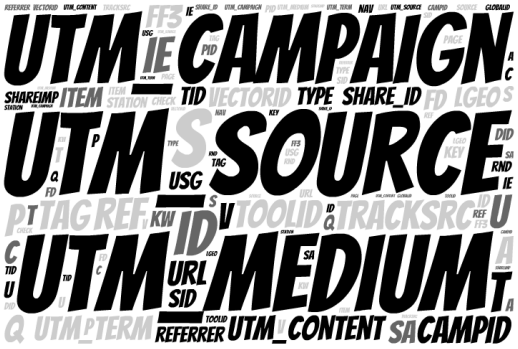

Fig. 2: Word cloud for common keys. Size indicates prevalence, with $\log_2$ applied to size weights for presentation

B. Common Query String Use-cases

The word cloud of Fig. 2 visualizes the most common keys in our data. Leading the way is key `utm_source` with 128.5 million instances; in 14% of all shortened addresses. That key – like 7 of the 10 most popular – and all those prefixed by "`utm`" are used to monitor referrers and traffic campaigns. The "Urchin tracking model" (UTM) is a structured means of link tracking that has become widespread due to its integration into the Google Analytics platform. In contrast, many keys are ambiguous in meaning and/or used by specific web platforms without an obvious naming convention. Single-letter keys are very common as are those that build around "`id`".

C. Privacy-sensitive Pairs

The bulk of query strings are uninteresting, serving as opaque identifiers of system objects or benign session parameters. More interesting are those that reveal personal information about the identity, location, network, *etc.* of whomever visited and subsequently shared a URL. At this point we do not concern ourselves with whether these sensitive key-value pairs are intended or crucial to page rendering, only that they are present within the URL.

Tab. I groups keys by the theme of information they reveal. Constructing this table involved manually inspection of the 861 keys with 100k+ occurrences. While key names are often indicative of their use, we also surveyed the values associated with those keys to confirm that sensitive data is present.[1] For example, we want to confirm that `zip` keys usually have 5-digit numerical values. With the exception of the "authentication" category, plaintext and human-readable values are the norm. Thus, Tab. I makes clear a tremendous amount of

THEME	KEYS	SUM-#
ALL URLS	—	892,934,790
URLS w/keys	*	490,227,789
referrer data	utm_source, ref, tracksrc, referrer, source, src, sentFrom, referralSource, referral_source	259,490,318
geo. location	my_lat, my_lon, zip, country, coordinate, hours_offset, address	5,961,565
network props.	ul_speed, dl_speed, network_name, mobile	3,824,398
online identity	uname, user_email, email, user_id, user, login_account_id	2,142,654
authentication	login_password, pwd	672,948
personal identity	name1, name2, gender	533,222
phone	phone	56,267

TABLE I: Keys w/100k+ occurrences having likely privacy ramifications, based on manual inspection[1]

personal information is potentially leaked via published URLs. In some ways this table *under reports* the risks. For example, the key "email" appears 103k times but there are 637k pairs where the key matches the pattern *email*. Further, there are 1.7 *million* email addresses in the corpus based on a pattern match over values, mapping to 2000+ unique key labels.

However, one must also be careful about such claims. There is no way of knowing to what extent values are "personal". Geographic coordinates in a URL could be referencing a user's exact location or, quite differently, be centering a map application over a landmark. We do not attempt to validate any of the mined personal information because of ethical considerations. This is particularly relevant when handling authentication credentials. Fortunately, when password or its analogues are present the values are *almost* always encrypted. Moreover, it seems that best practices are followed (*e.g.,* salting) as the MD5 and SHA hashes of 100 common passwords matched no values. Unfortunately, there are isolated examples (our shallow search found several dozen) of full credentials being passed in plain text via a query string. In the most egregious examples: (1) the credentials to an "admin" account were revealed, and (2) the user/password were for a site serving extremely personal information. These situations are "smoking gun" examples that would prove interesting to readers. Regrettably, the sensitivities surrounding these URLs are such that they cannot be published without significant redaction (*e.g.,* https://www.⊙.com/index.aspx?accountname=⊙ &username=⊙&password=⊙).

D. Value Entropy

The previous sections elucidate our position that query strings have privacy concerns, while demonstrating the problem is difficult to quantify. An online system to reduce such data leakage needs an algorithmic means to detect sensitive key-value pairs. Our work has discovered the *diversity* or *entropy* of a key's value set is helpful in this respect. For example, a key that is used in a binary fashion will have few unique values and low entropy. Even if this information were personal (*e.g.,* via the gender key) it does not reveal a terrible amount about the user in question. In contrast, high entropy keys have so many unique values that they can describe very specific properties towards identifying an individual.

Based on a diversity (d) calculation that lies on $(0, 1]$ we find that most of the keys we identify in Tab. I lie on $0.33 < d < 0.66$, in turn helping us find additional sensitive (but less popular) keys. Examples of these keys and their diversity metric include user (0.53), email (0.49), and my_lat + my_lon (both 0.38). We suspect analysis of the key distribution could also be insightful. Whereas the above examples are quite uniformly distributed, keys like utm_source follow a power-law distribution led by values twitterfeed (self-explanatory; 26% of all values) and share_petition (from change.org; 7.5% of values).

IV. Link Shortening Service

Discussion now shifts to CleanURL, our proposal to reduce privacy disclosures via URL query strings. CleanURL consists of back-end logic to determine the (1) necessity, and (2) sensitivity of key-value pairs (Sec. IV-A). The output of the system is a sanitized URL that is graphically presented to end-users for confirmation or modification of that result (Sec. IV-B).

A. Argument Removal Logic

When attempting to sanitize a URL there are two properties of each key-value pair to assess: its *sensitivity*, whether the value contains private data – and its *necessity*, whether the content of the page renders correctly if the pair is removed. As an example, consider pair zipcode=12345 which meets our sensitivity criteria. The exclusion of that pair might have 3 possible results: (1) The pair in no way affects page rendering. (2) The pair non-significantly affects rendering (*e.g.,* the zipcode was being used by a weather widget in the sidebar, orthogonal to main content). (3) The pair significantly breaks rendering (*e.g.,* the entire page previously displayed weather for the zipcode, but now redirects elsewhere). Under conditions 1 and 2 the pair would be stripped from the output URL, whereas situation 3 would issue a warning to the user. Non-sensitive and non-necessary pairs can also be stripped.

While simply stated, programmatic methods for sanitization logic are complex. With respect to *necessity* we consider:

- VISUAL DIFF: The two URLs (pair inclusive and exclusive) are rendered as down-scaled 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 and the delta size computed.

Both have proven moderately effective in preliminary testing. The primary complication is the presence of dynamic content (*e.g.,* advertisement images changing on every reload). This is one advantage of the HTML diff: although advertisements' visual impact may be significant, the code used to fetch them is often quite static. In practice one needs to select or learn diff thresholds which can tolerate small amounts of dynamic noise and well represent the degree of change.

[1]We do not contend that every value associated with these keys is privacy revealing. Instead we use Monte-Carlo sampling to confidently determine that at least a *majority* of values match an expected format.

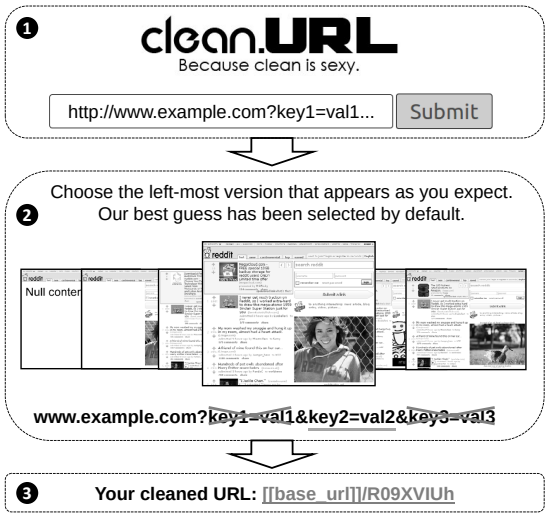

Fig. 3: Simplified screenshots detailing CleanURL interface

With respect to determining pair *sensitivity* we rely on:

- Regular expressions gleaned from the naming patterns of known sensitive keys. It is also possible to pattern match over structured sensitive values, *e.g.,* when email addresses are present.
- Mining URL corpora with metrics like key entropy, which can indicate sensitive pairs per Sec. III-D.
- Human feedback loops about output correctness per the CleanURL GUI (see next subsection). Note such feedback will only be available after public release.

B. CleanURL User Interface

Recall that for prototyping purposes we wrap our sanitization logic as a stand-alone link shortening service. A typical session with the shortener is depicted in Fig. 3. A user begins by entering a URL in a simple form field. This sets off the computational removal logic whereby all combinations of parameters from the query URL are enumerated. For each parameter combination the webpage source (*i.e.,* HTML) is downloaded, visually rendered, and input into our diff functions. Sensitivity properties are also investigated. Ultimately, combinations are sorted from most-to-least privacy preserving.

This ordering is the basis by which screenshots are presented in a "shuffle" selector to the end-user (see Fig. 3). The "suggested" version is selected by default: the combination that faithfully renders the page while removing the most sensitive parameters. If a sensitive parameter cannot be removed the end-user will be notified. Our design goal was to visualize the impact of URL manipulation and achieve end-user awareness while still maintaining a clean, simple, and usable interface. The user is free to browse/shuffle through all presented combinations. Both a sanitized and shortened URL is returned once a selection is confirmed. If our logic was too aggressive in removing parameters this interface gives the user a chance to correct that error. It also provides an opportunity to better understand human factors, collect ground-truth, and analyze the sensitivities surrounding certain data types.

V. Conclusion

In conclusion, we have analyzed 892 million user-submitted URLs, many of which are destined for public broadcast. We found that many of these links have query strings with key-value pairs containing sensitive data ranging from tracking identifiers (`utm_*`) to contact details (`email`), location information (`my_lat`, `my_lon`), and even passwords. These results motivate our position that the privacy impact of URL query strings should be analyzed further, especially given our prevalent Web 2.0 information sharing culture.

Our proposed CleanURL system addresses the privacy ramifications of URL query strings. Input URLs are programmatically analyzed to identify which key-value pairs lack *necessity* (do not affect page rendering) or are *sensitive* (reveal private information). CleanURL's eventual launch will permit usability measurements and leverage human feedback loops. Future work will also explore the impacts of REST-friendly URLs (where key-values are embedded along the URL path) and the notion that key sensitivity might vary as a function of the host domain. In total, we hope the evidence presented herein will encourage more dialogue and research into related Web 2.0 privacy issues.

Acknowledgements: Daniel Kim and Kevin Su are thanked for their contributions on a preliminary version of this work [13].

References

[1] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. P. Markatos, and T. Karagiannis, "We.B: The web of short URLs," in *WWW '11: Proc. of the 20th Intl. Conf. on World Wide Web*, 2011.

[2] B. Krishnamurthy and C. Wills, "Privacy diffusion on the web: A longitudinal perspective," in *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, 2009.

[3] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna, "Two years of short URLs Internet measurement: Security threats and countermeasures," in *WWW '13: Proceedings of the 22nd International Conference on World Wide Web*, 2013.

[4] S. Chhabra, A. Aggarwal, F. Benevenuto, and P. Kumaraguru, "Phi.sh/$ocial: The phishing landscape through short URLs," in *CEAS '11: Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, 2011.

[5] S. Lee and J. Kim, "Fluxing botnet command and control channels with URL shortening services," *Computer Comm.*, vol. 36, no. 3, Feb. 2013.

[6] B. Pfitzmann and M. Waidner, "Privacy in browser-based attribute exchange," in *WPES '02: Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2002.

[7] M. Balduzzi, C. Gimenez, D. Balzarotti, and E. Kirda, "Automated discovery of parameter pollution vulnerabilities in web applications," in *NDSS '11: Proc. of the Network and Dist. Sys. Security Sym.*, 2011.

[8] D. Weiss, "The security implications of URL shortening services," April 2009, http://unweary.com/2009/04/the-security-implications-of-url-shortening-services.html.

[9] F. Klien and M. Strohmaier, "Short links under attack: Geographical analysis of spam in a URL shortener network," in *HT '12: Proceedings of the 23rd ACM Conference on Hypertext and Social Media*, 2012.

[10] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: The underground on 140 characters or less," in *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010.

[11] J. Wilberding, A. Yates, M. Sherr, and W. Zhou, "Validating web content with Senser," in *ACSAC '13: Proceedings of the 29th Annual Computer Security Applications Conference*, December 2013.

[12] Z. Pehlivan, M. Ben-Saad, and S. Gançarski, "Vi-DIFF: Understanding web pages changes," in *DEXA '10: Proceedings of the 21st International Conference on Database and Expert Systems Applications*, 2010.

[13] D. Kim, K. Su, A. G. West, and A. J. Aviv, "CleanURL: A privacy aware link shortener," Univ. of Penn., Tech. Rep. MS-CIS-12-12, 2012.