Security and Privacy Implications of URL Shortening Services

Alexander Neumann, Johannes Barnickel, Ulrike Meyer
IT Security Group
RWTH Aachen University

Abstract

URL shortening services replace long URLs with shorter ones and subsequently redirect all requests for the shortened URL to the original long URL.

In this paper we discuss and empirically analyze security and privacy risks caused by the use of URL shortening services. We empirically determine the most popular URL shortening services currently used on Twitter and analyze these with respect to malicious behavior, user tracking, ease of enumeration, and leakage of URLs to search engines. Also, we introduce a new attack scenario to enable SSL-only circumvention using SSLStrip and shortened URLs. Finally, we empirically analyze the use of URL shortening services in more than 7 million spam e-mails collected over the past seven years and determine the spam detection performance for the most popular services found.

1 Introduction

With the advent of social networks and Twitter, on which messages are required to fit into 140 characters, compression of URLs becomes more and more crucial. In order to turn long URLs into shorter ones, so-called URL shortening services (**USS**) are offered by various entities. When users want to shorten a long URL, they submit the long URL to a shortening service that returns a short URL which typically does not exceed 30 characters. The users then include the short URL instead of the long URL e.g. in a tweet, short message or e-mail. When others request the short URL, the shortening service automatically redirects them to the original, longer URL. An example for a short URL generated by the shortening service `bit.ly` is `http://bit.ly/dLbCeC`, which consists of just 20 characters.

While the use of URL shortening service is very convenient, it imposes a number of risks for users submitting long URLs, users and machines requesting short URLs as well as the target servers hosting the resource to which a short URL points. Malicious or compromised USSes may redirect users of vulnerable browsers to websites containing malware. Innocent users may submit a meant-to-be secret URL to a shortening service, but the service may leak this short URL to search engines. Also, an attacker may enumerate the short URLs generated by a USS in order to find secret URLs. Users requesting short URLs from a USS may be tracked by this USS with the help of cookies. Also, they may unwittingly provide sensitive information to the USS by the information from the HTTP referrer included in their request for a short URL. Vulnerable servers may be attacked with the help of a USS by attackers submitting attack URLs to a USS that requests and checks the submitted URL before shortening it. URL shorting services may simply fail and thereby render the short URLs unusable. Finally, USSes may be used to enable SSL-only circumvention with SSLStrip.

In this paper we discuss these potential risks in more detail and empirically analyze popular USSes with respect to these risks. In particular, we empirically determine the currently most popular USSes used on Twitter with samples of messages containing up to 10% of all public Twitter messages sent during two periods of 24 hours each. We evaluate these USSes with respect to malicious behavior, user tracking, ease of enumeration of the generated short URLs, and leakage of secret URLs to search engines. In addition, we empirically determine the most popular USSes used in over 7 million spam messages collected in the past seven years. We analyze the spam detection performance for the most popular services found. Finally, we analyze how spammers obfuscate shortened URLs to hide their use.

2 Related Work

Shortly after the public release of the USS `tinyurl.com` in 2002, Thurman proposed the idea of enumerating shortened URLs [1] ("tinyurl whacking"). In this paper we pick up the idea and present the results of enumerating the ten most popular USSes used on Twitter in Section 4.2.1.

In 2008, McGrath and Gupta [2] found evidence that URL shortening services were being used in a set of 69,624 phishing e-mails, but the proportion of shortened URLs was very small. In this paper (Section 7) we present a similar analysis for 7,898,130 spam e-mails collected since October 2003. On top of this we analyze methods used by spammers to obfuscate shortened URL included in these email messages. In addition, we determine the most widely used USSes in these email messages and analyze the countermeasures the ten most most popular USSes use against spam.

In [3] Weiss discusses security implications of URL shortening services used on Twitter, mentioning several risks: The real destination is hidden, shortened URLs could be used to hide the real URL in spam e-mails, and shortening services can be compromised. While mentioning these risks, the author does not conduct any empirical analysis. Similarly, Schachter [4] analyzes the risks of shortening services, and mentions unreliability regarding

latency and long time availability, possibly hacked services, use of short URLs in spam, lower search engine rankings for shortened URLs, loss of information about traffic source for the target, and user tracking. We analyze these and new threats later in this paper.

In a study by Fetterly et al. in 2003 [5], the authors repeatedly access a set of 151 million URLs eleven times over a time of ten weeks. After one week, about 0.5 % of the URLs are not accessible any more. After ten weeks, this percentage rose to 3 % of the original set of URLs.

Grier et al. [6] analyze over 400 million public tweets on Twitter in 2010 to identify 2 million spam and phishing URLs. They use the click statistics generated by `bit.ly` for shortened URLs to measure a spam click rate of 0.31%. They conclude that spam on Twitter is far more successful than e-mail spam. However, they also find that more than 97 % of the shortened URLs they had collected from spam messages received no clicks at all, while the remaining 3 % are clicked very often.

In an article published in January 2010, Vilas [7] describes various additional methods for abusing USS's in the context of data storage, creating short links of JavaScript code, infinite redirection loops, and defeating a lookup service for shortened URLs [1] by building chains of shortened URLs. We do not consider these additional methods in this paper.

3 List of URL Shortening Services

To empirically analyze the risks identified in the next section, a list of URL shortening service is required. We start by collecting a list of public, general purpose USSes from different sources. As this list is too large to investigate all USSes found, we proceed with determining the most popular USSes used on Twitter. These USSes are then used for all further experiments described in this paper.

Several lists of shortening services have been found, e.g. in the Firefox add-on *ShortenURL*[2], the USSes supported by `longurl.org` [8], and several lists on blogs [9, 10, 11, 12]. More USSes were found by examination of host names of URLs found in spam e-mails (see Section 7.1).

The data from these sources combined contains 610 unique host names of USSes. However, this list S_{all} also includes several special purpose shortening services (e.g. `amzn.to`, `fb.me`), which cannot be used to shorten arbitrary URLs. As we examine security implications of USSes available to the public, we deleted the special purpose shortening services from the overall list, leaving us with a list S_{USS} of 527 public, general purpose USSes.

Twitter provides an HTTP based streaming API [13] which delivers new messages matching search criteria or a small percentage of all newly posted messages. The function for the latter is called **`statuses/sample`**. Each API call requires authentication with a valid Twitter account. The percentage of all public messages delivered by this API function depends on the account's access role. We used an account with the access role "Gardenhose" that receives up to 10 % of all public messages. We collected public messages from Twitter during two periods of 24 hours each. We chose these two periods to cover different usage patterns during regular weekdays as well as on weekends. In the first period between Thu, 28 October 2010, 00:23 CET and Fri, 29 October 2010, 00:23 CET we sampled 7,547,787 messages. In the second period between Sat, 8 January 2011, 13:19 CET and Sun, 9 January 2011, 13:19 CET we sampled 8,691,168 messages. We extracted 1,208,862 unique URLs from the first sample and 1,143,838 unique ones from the second sample. We parsed these URLs for host parts included in the list S_{USS} defined above to discover the most popular USSes.

Table 1 shows the number of short URLs found in the first and second sample for the top 20 host parts. Note that as `j.mp` is an alias for `bit.ly`, these are also the top 19 USSes. The service `bit.ly` (including `j.mp`) clearly dominates the list of USS in both samples followed by the services `t.co`, `tinyurl.com`, `goo.gl`, and `ow.ly`. The top ten shortening services cover 97 % of all shortened URLs in the first, and 96 % in the second sample.

Host	URLs (first)	Percentage	URLs (second)	Percentage
bit.ly	347,565	62.70	255,413	59.17
t.co	40,801	7.36	42,727	9.90
tinyurl.com	34,962	6.31	31,387	7.27
goo.gl	31,259	5.64	25,044	5.80
ow.ly	30,002	5.41	14,711	3.41
dlvr.it	18,368	3.31	16,917	3.92
is.gd	10,704	1.93	9,056	2.10
j.mp	8,988	1.62	9,117	2.11
migre.me	5,464	0.99	3,179	0.74
dld.bz	4,252	0.77	3,899	0.90
lnk.ms	3,346	0.60	2,822	0.65
wp.me	2,926	0.53	2,107	0.49
tiny.ly	1,728	0.31	2,790	0.65
twurl.nl	2,200	0.40	1,683	0.39
su.pr	1,828	0.33	1,397	0.32
3.ly	369	0.07	2,825	0.65
post.ly	1,703	0.31	1,089	0.25
tiny.cc	1,515	0.27	647	0.15
durl.me	485	0.09	426	0.10
adf.ly	329	0.06	558	0.13

Table 1: Top 20 USSes found in Twitter samples

4 Risks for Users

The use of shortening services imposes risks on users submitting URLS as well as users requesting shortened URLs. These threats are discussed in the following.

4.1 Sensitive Information: HTTP Referrers

When sending an HTTP request for a shortened URL to the shortening service, the user's browser transmits the

[1] `http://longurl.org`
[2] `http://j.mp/g3pWGp`

HTTP referrer as one of the HTTP header fields to the shortening service. The user's browser typically sets the HTTP referrer to the URL of the web page on which the link to the short URL was clicked. This enables the shortening service to conclude which web page contains links to the requested short URL. Some shortening services (e.g. `nvg8.it`) provide this information to the creator of the shortened URL. While it is already possible to use web search engines to find hyperlinks from public websites, this referral transmission to USSes may expose otherwise hidden web pages, e.g. private collections of bookmarked links, which are publicly accessible but neither linked from anywhere nor indexed by search engines. One may argue that the HTTP referrer is always transmitted to the target web server anyway. However, when accessed via the shortened URL, one or two additional parties gain access to this information: The USS and (for some services) the creator of the short URL.

4.2 Sensitive Information: Submitting Secret URLs

Often, users who submit URLs to a USS are not aware of the fact that once the URLs are submitted to the USS, these URLs are not secret any more. At least the administrators of the service always have access to the URLs. Furthermore, many USSes provide a list of popular or recently shortened URLs to the public, where they can be found by search engine crawlers, e.g. `ke-we.net`. USSes might even submit URLs to search engines directly. The experiment outlined in Section 4.2.2 determines who accesses URLs submitted to shortening services and which services leak these URLs to search engines.

Some shortening services do not even have a clear privacy statement on what the service does with submitted URLs and collected data (e.g. `tinyurl.com`). In addition, many services do not describe any process or at least provide a contact e-mail address for removing a shortened URL such that users cannot withdraw secret URLs once they become aware of the leakage.

Another way how secret URLs can become known to unauthorized entities is enumeration. An attacker can try to enumerate every possible shortened URL of a service, request them, and thus gain knowledge of the target URLs. In the following, we describe an experiment in which we enumerate short URLs and search for secret URLs users have shortened.

4.2.1 Enumerating Shortened URLs

All shortened URLs from the top 10 shortening services on Twitter found in Section 3 have the following structure:

```
http://<host>/<path>
```

The `<host>` part is the USS's host name and `<path>` is the string uniquely identifying the short URL, called *tag* in the following.

Character Frequency Analysis: Attackers who are searching for secret URLs are primarily interested in URLs which have just recently been shortened, because the destination URL of older shortened URLs might not exist any more. For the attackers it is therefore interesting to know the structure of a target service's shortened URLs. If all the shortened URLs a service hands out in the last 24 hours start with the same letter, it might be a good idea to start enumerating URLs with this prefix.

We reuse the 16,238,955 twitter messages collected as described in Section 3. As most Twitter clients automatically shorten URLs contained in a message before posting it, one can assume that many of these short URLs were recently generated. We study the structure of new unique tags for the ten most popular services by extracting all short URLs of a particular service from the twitter messages and analyze the frequency of characters in the unique tags. By comparing tags found in the URL lists for the two sampling periods, the structure of newly generated unique tags can be researched.

The unique tags of shortened URLs from `bit.ly` are composed of six upper- and lower-case letters and numbers. The service seems to choose the first character incrementally, because the characters `9` and `a` to `d` each have a much higher frequency of 13 to 20 % than the surrounding characters, which only have a frequency of 1 % or below. All characters at positions two to six are uniformly distributed across the complete range which leads to a relative frequency of 1.6 %. This leads to the conjecture that `bit.ly` chooses the first letter of a shortened URL's unique tag sequentially.

The unique tags from service `t.co` have a structure which is similar to the tags of `bit.ly`: Seven characters upper- and lower-case letters and numbers. Each character has a relative frequency of 1.6 % regardless of the position in the unique tag. This is the expected value if each character of the unique tag is chosen independent from the uniformly distributed set of all characters and numbers.

Unique tags for the USS `tinyurl.com`-URLs have an interesting structure. The service does not distinguish between upper- and lower-case tags. The tag is composed of seven lower-case letters and numbers. In both samples, most tags begin with `2` (78 %) or `3` (20 %). The only other character which has a relative frequency worth mentioning is `y` with 1.7 %. The second character is very likely to be `3` to `9` or `a` with a relative frequency of 7 % each. The characters at positions three to seven all have a relative frequency of 3 %, which is expected for a uniformly distributed set of 33 characters (25 letters plus 8 numbers).

It is interesting that the letter `i` and the numbers `0` and `1` are found only very rarely, regardless of the position. One can speculate that this might be because an upper-case `i` can easily be misread as `1`.

No obvious structure can be observed for goo.gl, the characters seem to be chosen randomly and independently from a uniformly distributed set of upper- and lower-case letters and numbers.

The services ow.ly, is.gd and migre.me seem to choose the characters for the unique tags sequentially. The same holds for the services dlvr.it, dld.bz and lnk.ms, with the exception that each of these services omits some characters.

Enumeration Process: Based on the preceding character frequency analysis, the services and shortened URL-ranges have been selected for enumeration. This leads to a number of 233,000–238,000 URLs per service, which we requested. As the service goo.gl enforced a request limit and stopped delivering redirects when many shortened URLs are requested too fast, we introduced a delay of 100ms after each request. With these limits in place, enumerating the shortened URLs for goo.gl took 27 hours.

Results: All shortened URLs selected in the previous section are requested. In all requests we use a User-Agent string simulating the Firefox web browser running under Linux.

We evaluated the target URLs manually and found a lot of sensitive information. This includes 153 web-based administrative interfaces. These are URLs that contain the string "/admin/" and return the HTTP status 200. Filtering the target URLs for the host name docs.google.com searching for the string "authkey=" and HTTP response code 200, we found 71 publicly accessibly documents hosted by Google Documents. These documents are allowed to be accessed by every person knowing the URL (with the included authkey) and are usually not indexed by search engines. Manual inspection leads to several archives of private photos, a seminar paper, a treasurer's report for a company, two curriculum vitae and the list of names, addresses, and telephone numbers of a kindergarten.

In summary, we have shown that enumerating USSes is feasible and that sensitive or private information can be found this way. Obviously users are not aware of the fact that the URLs can be revealed by USSes to others.

4.2.2 Submitting Secret URLs

This section describes an experiment for testing whether URLs submitted to shortening services are leaked by them, e.g. to search engines.

Preparations: In this experiment, the domain fd0.me has been used for all generated URLs. A web server is configured which logs all requests to URLs where the host is either fd0.me or www.fd0.me. The server responds to all URLs for the root page ("/") with a generic information page that links to the description of this research project. Furthermore, the page contains a contact e-mail address. When a URL is requested and the path component contains at least three forward slash characters ("/") a special web page is returned. This page is titled "Secret Webpage" and contains data from the request such as the requested path, protocol, the request method and the User-Agent string. Furthermore a string called "Secret Tag", general information on the experiment, and a contact e-mail address are listed. The secret tag is constant, the string knycsUpjaj8 is returned for each request regardless of the URL's path component. This string is unique and had no hits when searched for in a search engine like Google before the start of the experiment.

Submitting URLs: For each USS, two URLs are generated using the following templates:

- http://fd0.me/secret/SERVICE/TIMESTAMP
- http://www.fd0.me/blog/archive/2011/01/14/index.php?article=SERVICE#TIMESTAMP

The string SERVICE is replaced by a unique eight character string for each service and TIMESTAMP by another eight character string uniquely identifying the time and date when this URL was submitted to the service. The first URL template should catch every person's eye and evoke curiosity so that the person clicks on the URL. URLs generated using this template are termed *secret* URLs in the following. All URLs generated after the second template should appear harmless and not very interesting, they are called *blog* URLs. Examples for both types of URLs are:

- http://fd0.me/secret/a0df29ac/bb42ce8b
- http://www.fd0.me/blog/archive/2011/01/14/index.php?article=69e325eb#a5a6c61c

The secret URLs have been manually submitted to as many shortening services as possible. All services from S_{USS} have been tried, but only those which were functional and allowed anonymous shortening without prior registration were used. URLs are submitted only once to services which operate several different host names (e.g. bit.ly and j.mp belong to the same service). For this experiment, 326 secret and 208 blog URLs have been submitted to 225 services.

Results: The web server's log file has been analyzed 4 weeks later. It contained 687 requests for 112 secret and 68 blog URLs. All requests which had occurred within 60 seconds after submission are assumed to be automatic requests from the shortening service. This leaves 497 requests for 49 secret and 31 blog URLs which had occurred afterwards. URLs generated for 42 different services were accessed after 60 seconds, i.e. 19 % of all tested services.

The resulting log file entries are examined manually regarding an HTTP referrer. Through this analysis, nine URLs of non-public web-based administrative interfaces for the services are discovered, so at least for nine services an administrator checked newly submitted links manually. Furthermore, at the time of writing, the administrators of

four other shortening services, who had seen the secret URLs and the contact information supplied, contacted one of the authors of this paper via e-mail and asked for details. They all stated that their respective shortening service is small and regularly used by spammers to shorten malicious URLs, therefore they check "suspicious" URLs manually. All in all, thirteen administrators of shortening services had a look at the submitted URLs.

When analyzing the `User-Agent` strings sent in the requests, we found the three search engines Google, Yahoo, and Baidu. We queried the search engines about the unique string `knycsUpjaj8` embedded in the response page for each generated URL. Only Google delivered the right number of hits, Baidu and Yahoo (international) did not return any results. The German Yahoo search returns one result.

In summary, the results of this experiment demonstrate that URLs submitted to shortening services do not remain private. Several services leaked the URLs to search engines, or the administrators checked the URLs manually.

4.3 Nonworking Shortening Service

URL shortening services have a history of failure. Services silently stop working, like the service on the Tonga top level domain `to`, or announce discontinuance of business. One of the first shortening services, `makeashorterlink.com`, announced the termination of the service, but could be acquired by `tinyurl.com` which continued the service.

Users relying on shortening services may leave their readers with nonworking URLs, for example when authors publish shortened URLs in books and other printed publications. This phenomenon ("link rot") is well known in the web and academic communities [14]. For citations in academic literature, a workaround has been proposed in [15].

The use of URL shortening services increases the risk that URLs stop working at some point in the future. This can be tolerated for short-living communication (e.g. Twitter), but is unacceptable for long-term use in books or academic references.

4.4 Hacked Shortening Service

When a shortening service is hacked, two different threats emerge to users of that service: Shortened links may stop working and requests can be redirected to advertising websites and/or websites containing malware. This also poses a risk to their web browser and operating system. Shortening service hacks have been observed in the wild already. For example, in June 2009, the service `cli.gs` got hacked. According to the service's blog, 2.2 million URLs got redirected [16] to a different destination and 7% could not even be restored from the backup and are permanently lost [17].

4.5 Tracking Users

A shortening service can extract much data from the request for shortened URLs. This data includes the requested URL, time of request, requesting browser version, HTTP referrer, and the IP address of the user's machine. The latter allows for discovering different Internet connections of a user and often also the user's physical location using a Geo IP database.

In addition, like other web servers, many shortening services set a unique cookie in the user's browser. The browser then transmit the cookie to the service every time a short URL is requested. While identifying users is not necessary for the shortening service to function, setting a unique cookie enables the service to track the user without much effort. When the validity period of the unique cookie is long enough, the service can over time build a history of accessed URLs and mine interesting data.

4.5.1 Cookie Analysis

Cookies are normally used to recognize an HTTP client over multiple HTTP requests. In the case of URL shortening services, this is not necessary, as every request for a shortened URL should be independent of all other requests executed before. Being able to recognize a user over a larger time span enables the service to aggregate interesting data on the user's behavior and interests.

The set of shortening services found in both Twitter samples as described in Section 3 is used for this experiment. Where available, we randomly selected two distinct URLs for each service from the Twitter samples. This gives us a list of 319 unique URLs for 187 different shortening services.

Each URL from this list is requested 83 times, each time by simulating a different web browser. From the web server's response, the values of the `Location` and `Set-Cookie` headers are stored in a database. The `Location` header values will be analyzed in Section 5.1. The `Set-Cookie` header instructs browsers to store HTTP cookies. 36 of the 84 cookies found are session cookies, the remaining 48 are persistent. The USS can therefore identify and track users, even when their IP addresses change.

For the cookies of each service, the quotient Q is calculated by dividing the number of unique values by the number of HTTP responses which try to set a cookie with this name. This value visualizes how "unique" the values for a cookie are. If Q is close to 1, many different values have been received, therefore the cookie's value is more specific to a request. On the other hand if Q is small, only very few different cookie values have been received. This can suggest a generic configuration cookie, e.g. a cookie which stores a boolean value, but this is not always the case. For example consider a hypothetical service which stores the IP address of the user in a persistent

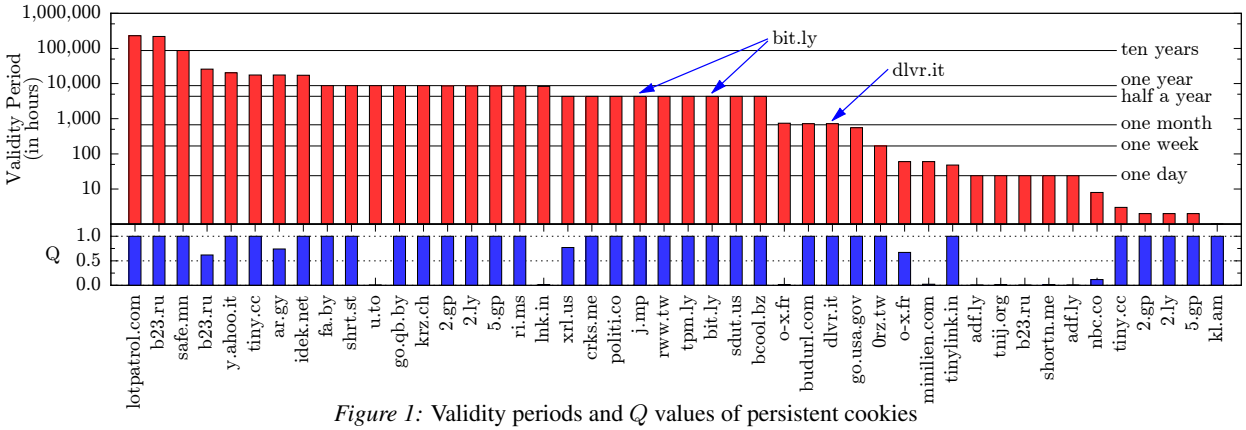

Figure 1: Validity periods and Q values of persistent cookies

cookie on the first visit. In this case the service is able to track the user over the validity period of the cookie, but the Q value for this cookie would be close to 0, because just one unique value is observed.

For example the cookie "u" from the USS `b23.ru` is set to the static value 1 in each HTTP response received. This cookie cannot be used to track a user and the Q value for this cookie is 0.059. On the other hand, the cookie "fwsid" of the same service is always set to a different value, a string of 32 characters, and has a Q value of 1. It is therefore very well suited to identify a user.

4.5.2 Results

It is found that 65 out of 187 tested shortening services use cookies, which is 35 %. Persistent cookies are set by 38 services. Figure 1 visualizes the validity periods and Q values of all persistent cookies. It shows that 28 services (or 15 %) set cookies with a validity period of half a year or more. Two services, `kl.am` and `tiny.cc`, tried to set two cookies with different names, but same validity periods and Q values. These cookies have been merged in Figure 1, so that for each service only one entry is included in the figure.

The services `bit.ly` and `dlvr.it`, that are found to set a persistent cookie in this section, also appear in the top ten list of popular shortening services on Twitter (see Section 3). The services are marked with arrows in Figure 1. The median of the validity periods is 180 days, which is nearly half a year.

Of all tested shortening services, more than one third use cookies and 15 % employ persistent cookies which allow tracking users over a period of 6 months or more. This is not acceptable, because cookies are not necessary for the operation of a shortening service. Using unique cookies which are valid over such a long time is at least questionable.

5 Risks for the User's Machine

This section outlines risks for the operating system of users' machines. We first describe the potential behavior of a malicious service. Then we describe an attack that allows for the circumvention of enforced SSL encryption of websites with the help of a URL shortening service.

5.1 Browser Attacks

In the preceding section we already mentioned that with the request for a URL, additional data is sent to the shortening service, including the version of the requesting browser. Attackers could set up an attractive USS which initially performs well and honestly. After establishing the service, the malicious USS could handle requests by browsers without any known security vulnerabilities normally while redirecting browsers with known vulnerabilities to malicious websites that exploit the version of the requesting browser. The attackers could sell visits from vulnerable browsers to botnet operators seeking new systems for their malware. Advanced users would not notice the malicious behavior, as they rarely use old and vulnerable browsers.

In the following we describe an experiment which tests how current services react when users access them with different web browsers.

5.1.1 Simulating Browser Versions

This section analysis the values received in the `Location` headers collected in the experiment described in Section 4.5.1.

When a URL is requested, the browser transmits its own name and version in the HTTP header `User-Agent`. We use this header to simulate different browsers and operating systems. We use the recent configuration file `http://techpatterns.com/forums/about304.html`[3]. This is a list of 83 different browsers and browser versions. The descriptions of all browsers

[3]Downloaded from `http://j.mp/cFh7Xd`

included in this list can be found on our website.

For each User-Agent string we sequentially requested all 319 URLs once between January 26th 19:00 CET and January 27th 03:00 CET in 2011 and stored the responses in a database. For each response, we stored: HTTP response code, Location response header value, and Set-Cookie response header value. If no valid HTTP response could be read after 10 seconds or the service was not reachable, the query was ignored in the analysis. We then analyzed if the HTTP response code and the value of the Location header differ when queried with different User-Agent strings.

5.1.2 Results

First the number of different combinations of the HTTP status code and the value of the Location header in all HTTP responses was calculated for each service. We found that of the 319 unique URLs, 292 always return the same HTTP status code and Location header regardless of the User-Agent header value. For these services, no malicious behavior could be observed. For 11 URLs, there are two different combinations of status code and Location header value. The remaining 16 URLs were not reachable.

The USS sn.im can also be reached under the host names snipr.com, snipurl.com, and snurl.com. The service normally returns HTTP status code 301, but between 01:31 and 01:37 o'clock CET on 27 January 2011 it returned the status code 410 for all URLs. As this could not be reproduced later for the browser strings used in requests at that time, it is assumed that this behavior originated in a technical defect at the shortening service. Therefore, no malicious behavior could be observed.

The short URLs of USS su.pr usually redirect directly to the target URL, but sometimes requests are redirect to another host name of the service (stumbleupon.com), which delivers the target URL in an HTML frameset. We do not known what criterion selects which behavior, but this clearly is not malicious.

to.ly usually redirects to the destination URL with the HTTP status code 301. However, at 00:09:41 CET on January 27th 2011, it returned an HTTP status code 503 for two requests, which could not be reproduced. We assume that it is due to a technical failure. No other malicious behavior could be observed for this service.

The most interesting result from this experiment are the responses for the only URL http://ri.ms/5wgv6 of the service ri.ms, where we observed 83 different combinations of HTTP status code and Location header. This host name belongs to the service tinyarro.ws. For 82 requests, the service returned HTTP status 302 and a Location header that points back to a preview page at tinyarro.ws. Included in this URL is the original target URL of the shortened URL and a counter which increments on each request. The different values for the counter are the cause for the huge number of different Location values. One example URL (wrapped to increase readability) is:

```
http://tinyarro.ws/preview.php?page=http
%3A%2F%2Fwww.cin.hdmais.com.br%2Fcomponent
%2Fcontent%2Farticle%2F&count=248
```

When accessed, a preview page for the target URL is shown together with a countdown timer implemented in JavaScript, which redirects to the target URL after ten seconds. The other 3 requests result in a response with HTTP status 301 and a Location header which directly points to the target URL http://www.cin.hdmais.com.br/component/content/article/.

On investigating this behavior it is found that HTTP status 301 is only returned when the User-Agent HTTP header is set to the browser version string of the browser Dillo (http://www.dillo.org/) in version 2.0, and for the empty string. Due to an error, each URL has also been requested with an empty User-Agent header and these requests also return HTTP status 301. This behavior could be reproduced, for each request with obscure, invalid or empty User-Agent HTTP header, the status 301 is returned and the Location header contains the target URL. It is assumed that this behavior had been implemented to only redirect browsers that are known to parse and execute JavaScript to the preview page with the countdown timer and redirect all other HTTP clients directly to the target URL. This behavior is not considered malicious, but it clearly shows that shortening services exist, which handle different HTTP clients in different ways.

In summary, the experiment did not reveal any services which show malicious behavior, but evidence could be found that at least one service returns different responses when requested with different browsers.

5.2 SSL-Only Circumvention

In the following we describe how URL shortening services which operate over plain HTTP[4] can be used by an attacker to trick users into submitting their passwords in plain text over the network without using HTTPS. The attacker achieves this with the help of SSLStrip[5].

SSLStrip is a program by M. Marlinspike which can act as a proxy between users and the real website. The proxy intercepts all HTTP requests and rewrites all HTTPS URLs to HTTP within the responses from the server before forwarding the response to the users' browsers. This way, the users do not encrypt their traffic to the proxy, but the proxy itself can use an encrypted connection to the real website.

For the attack we assume a man-in-the-middle attacker

[4]Note that while a few services like bit.ly offer HTTPS, most USSes such as tinyurl.com and Twitter's t.co do not.

[5]http://j.mp/5FZiM, http://j.mp/e6g9NG

that is able to read and intercept any packet victim user sends to a service. In particular, the attacker can read, intercept, and manipulate HTTP requests, but is not able to decrypt and read HTTPS connections. We further more assume that the user (Alice) uses a public messaging service *LoudSpeaker*, a social network *Badaboom* and a shortening service `sho.rt`. *LoudSpeaker* and *Badaboom* are only accessible via HTTPS and the HTTP cookies set by the services all have the "Secure" attribute so that the cookies are not sent in an unencrypted HTTP request. The shortening service `sho.rt` does only support HTTP. In this situation, the attacker's goal is to gain knowledge of the user's password on *Badaboom*.

The attacker can reach this goal with the following steps

1. Alice reads her *LoudSpeaker* message feed
2. She receives a message: "Hey, check out this group over at Badaboom: `http://sho.rt/xafAb`!"
3. Alice clicks on the link, her web browser sends an HTTP request for the URL `http://sho.rt/xafAb`. This short URL points to the HTTPS URL `https://badaboom.rt/groups/23`.
4. Using SSLStrip the attacker intercepts the HTTP request, executes the request on behalf of the user, replaces "`https://`" by "`http://`" in the response and serves the modified response to Alice's web browser.
5. Alice's browser follows the redirect and requests `http://badaboom.rt/groups/23`. All cookies set by *Badaboom* are marked "Secure", so none is included in this unencrypted HTTP request.
6. Using SSLStrip, the attacker intercepts the HTTP request, carries out an HTTPS request in the background, and serves the result to Alice's web browser.
7. Alice sees the regular Login page for *Badaboom* as no cookie has been sent in the request. She enters her username and password and clicks on the submit button.
8. Using SSLStrip, the attacker intercepts the HTTP request, stores the username and password for later use and carries out a HTTPS request in the background. In the response, the "Secure" attribute is removed from the cookies and all HTTPS URLs are again replaced by HTTP URLs, afterwards the modified response is served to Alice's web browser.

Experienced users are able to detect this attack by inspecting the URL or by wondering why they are requested to log in again, but inexperienced users are not. No HTTPS certificate is used, but also no error message is shown and the service functions. This is sufficient to fool most users.

This attack could be mitigated or even completely prevented when the web server offers HTTP headers according to the "HTTP Strict Transport Security" (HSTS or STS) standard and the user's browser has interpreted the header. At the time of writing, HSTS is an RFC draft.

6 Risks for Servers

In this section, risks for the operating system and other software running on servers are evaluated. This does not include the web servers running the shortening service, because these are exposed to the same threats as any other web application server.

6.1 Secret URLs

In addition to the risks for users introduced in Section 4.2, secret URLs submitted to shortening services might also pose a risk for the target server if they e.g. point to an administrative web-based interface. When this URL is submitted to a shortening service and that particular service publishes the list of the last twenty shortened URLs, attackers might just stumble upon the shortened URL. Furthermore, search engines can crawl the USS pages and add the secret URL to their index. Attackers might search for administrative interfaces or just for vulnerable versions of "phpMyAdmin" by using search engines. This technique is also known as *Google Hacking*.

6.2 One-Click Exploits and Denial-of-Service

Several URL shortening services test submitted URLs before returning a shortened URL. This procedure is usually employed to enforce the terms of service, in which many shortening services forbid submitting URLs pointing to other shortening services. The target URL is requested once and the HTTP response code is inspected.

This can be used to attack other servers. When a web server is vulnerable and the vulnerability can be exploited with just one HTTP `GET` request, it is sufficient for an attacker to submit a URL that exploits the vulnerability to a shortening service. This service will then request the URL from the server and trigger the exploit. Especially when one URL takes down the server, this approach can be very effective.

7 Shortened URLs in Spam

We use data obtained from a spam honeypot[6] running continuously since October 2003 to analyze the use of USS in spam. The honeypot consists of a website generating and displaying a new e-mail address each time it is requested, and a database to store the messages received. At the time of writing, the project was running for seven years and had collected 7,898,130 spam e-mails addressed to 5,594 different e-mail addresses (each corresponding to one e-mail address crawler visit). As no address has ever been used for real e-mail communication, all received e-mails are spam.

[6] `http://koeln.ccc.de/schnucki`

7.1 Parsing Spam

A Ruby script is used to parse the e-mail messages. The results are stored in a database. For each e-mail, the recipient e-mail addresses, date and time the e-mail has been received, and all URLs found in all attachments of content types `text/plain` and `text/html` are stored in the database. Attachments are decoded if a content-transfer-encoding (e.g. `quoted-printable` or `base64`) has been used and searched for strings which start with "http://" or "https://" or which contain "www". Before inserting the strings into the database as URLs, the script tries to parse each URL using the Ruby `URI::HTTP` class. Afterwards, valid URLs are requested and the results are stored in the database.

The complete process of parsing all 7,898,130 e-mails and searching for URLs took 52 days on a machine with a quad-core CPU (Intel Xeon, 2.33GHz) and 4GB of RAM running Linux. An amount of 12,831,780 URLs were found within the e-mails, of which 21,843 URLs have been marked as invalid (i.e. could not correctly be parsed as a URL), leaving 12,809,937 valid URLs.

We search the set of URLs for new USSes by first selecting all URLs whose response HTTP status code is in the range of 200 to 399, and afterwards matching the URLs using a regular expression. In this regular expression the length of the host name is in the range of 2 to 15 characters and the length of the path is in the range of 1 to 10 characters. Afterwards all host parts from the resulting list of URLs are checked by hand for shortening services. This way, 58 additional services are discovered which are not included in any of the other lists mentioned in Section 3. Searching for shortening services in all URLs found in the spam e-mails by matching the URLs against the set S_{all} leads to a list of 35,647 shortened URLs, which is 0.3 % of all 12,809,937 valid URLs. The top twenty services are listed in Table 2.

Host	Unique URLs	# Messages
xs.to	22,711	27,382
bit.ly	6,372	9,068
tiny.cc	1,133	1,308
hurl.me	1,109	1,969
urlpass.com	683	1,012
snipurl.com	338	534
migre.me	251	568
su.pr	231	1,415
snipr.com	230	344
snurl.com	211	174
is.gd	197	283
redir.ec	169	333
tinyurl.com	163	430
dwarfurl.com	105	77
moourl.com	100	95
uforgot.me	93	93
good.ly	83	88
9mp.com	82	91
urlme.in	68	97
qurl.com	60	80

Table 2: Top 20 USS found in spam e-mails

Service	Spam Detection Rate
`is.gd`	100.00 %
`tinyurl.com`	97.01 %
`snipurl.com`	63.67 %
`bit.ly`	57.00 %
`moourl.com`	57.00 %
`su.pr`	33.04 %
`migre.me`	4.38 %
`tiny.cc`	5.32 %
`redir.ec`	0.00 %
`urlpass.com`	0.00 %

Table 3: Detection rates of popular shortening services found in spam e-mails

7.2 USS Spam Detection Performance

In this section, we analyze the 16 services for which at least 100 shortened URLs have been found in spam regarding their spam detection rate. These are: `xs.to`, `bit.ly`, `tiny.cc`, `hurl.me`, `urlpass.com`, `snipurl.com`, `migre.me`, `su.pr`, `snipr.com`, `snurl.com`, `is.gd`, `redir.ec`, `tinyurl.com`, `9mp.com`, `dwarfurl.com`, and `moourl.com`. The results are summarized in Table 3.

All requests for URLs of `xs.to` returned HTTP status code 404: no URL was redirecting properly at the time of writing. It is therefore not possible to estimate the spam detection rate for `xs.to`.

The service `bit.ly` returns HTTP status code 301 for working and status 302 for disabled shortened URLs, where users are redirected to a warning page. It can therefore be concluded that `bit.ly` has a spam detection rate of 3,632 out of 6,366, or 57 %.

Most shortened URLs of the service `tiny.cc` are working. Only a single URL `http://tiny.cc/lsc3z` could be found that has been marked by the service as abuse. This is done by redirecting users to an abuse page. There are 60 other URLs which redirect to a "not found" page. Assuming that "not found" means the shortened URL has been deleted because of spam, this leads to a spam detection rate of 60 out of 1,128 URLs, or about 5.32 %.

The services `urlpass.com` and `redir.ec` did not recognize any URLs found in our honeypot as spam.

The service `snipurl.com` had two alias names in the list: `snipr.com` and `snurl.com`. The service returns the HTTP status code 301 for working redirections and HTTP status code 410 ("Gone") for disabled or deleted URLs, where a web page is shown which states that the shortened URL has been deleted. The spam detection rate is 496 out of 779, or 64 %.

11 of the 251 URLs found for `migre.me` redirected back to the server `migre.me` and the resulting web page showed a message that the requested shortened URL was

blocked. This makes a spam detection rate of 4.38 %.

The service su.pr returned HTTP status code 404 77 out of 233 times. Assuming that these URLs were deleted because of spam, this leads to a detection rate of 33.04 %.

All URLs of the service is.gd return HTTP status code 200. Users trying to access the shortened URLs are displayed a message which states that the URL is disabled due to a violation of the service's terms and conditions. Therefore the spam detection rate is 100 %.

The service tinyurl.com returns HTTP status code 301 for working shortened URLs. All 29 URLs which return status code 404 are probably invalid URLs because the format differs significantly from the usual one. The other 130 URLs which return HTTP status code 302 redirect to an abuse error web page. The spam detection rate of tinyurl.com is 130 out of 134 URLs, or 97 %.

The USS hurl.me was not operating properly, at least for all URLs found in spam e-mails. dwarfurl.com and 9mp.com were also inoperative. Therefore we could not determine their spam detection rates.

Within the set of 100 URLs of the service moourl.com 42 redirected to a page stating that the URL was disabled, and for 15 more a message was shown that the shortened URL could not be found. As these 15 URLs look valid, it is assumed that the deletion happened on purpose. This gives a detection rate of 57 %.

In summary, we show that URL shortening services are used in spam and the spam detection rates for most of the services are not optimal.

8 Conclusion

We have identified and examined how URL shortening services may introduce security and privacy risks. According to our analysis, none of the currently most popular URL shortening service exhibits malicious behavior. We show, however, that many of these shortening services are well-prepared for user tracking. Also, we show that by enumerating shortening services a lot of sensitive or private information can be found and several shortening services do leak submitted URLs to search engines. Future Work may include similar analysis of one click image hosting services or one click hosting in general. A monitoring service for USSes could be established, to verify the continuous performance of USSes regarding availability and spam detection. An extended version of this paper will be made available that contains all lists generated in our experiments.

References

[1] Thomas Thurman. Tinyurl whacking. http://thomasthurman.org/tinyurl-whacking, 2002.

[2] K. McGrath and M. Gupta. Behind phishing: an examination of phisher modi operandi. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, 2008*. USENIX Association.

[3] David Weiss. The Security Implications of URL Shortening Services. http://j.mp/b4bjO, April 2009.

[4] Joshua Schachter. on URL shorteners. http://j.mp/BJOml, April 2009.

[5] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. *Proceedings of the 12th international conference on World Wide Web*, page 97ff, May 2003.

[6] C. Grier, T. Kurt, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security 2010*, pages 27–37.

[7] Mario Vilas. Having fun with URL shorteners. http://j.mp/i2jexh, January 2010.

[8] URL Shortening Services - A List of URL Shorteners. http://longurl.org/services.

[9] Long list of URL shorteners. http://j.mp/y0ENQ, June 2008.

[10] Mike Koss. Bit.ly Dominates TinyURL and 213 other Link Shorteners. http://j.mp/1UFvr4, August 2009.

[11] Mike Koss. Exhaustive List of URL Shorteners. http://j.mp/axq6Oa, April 2009.

[12] Palin Ningthoujam. URL Toolbox: 90+ URL Shortening Services. http://j.mp/91Dq97.

[13] Twitter Inc. Twitter Streaming API: Concepts. http://j.mp/dkaT3J.

[14] R. Dellavalle, E. Hester, L. Heilig, A. Drake, J. Kuntzman, M. Graber, and L. Schilling. Going, Going, Gone: Lost Internet References. *Science*, 302(5646):787–788, 2003.

[15] Gunther Eysenbach and Mathieu Trudel. Going, going, still there: Using the webcite service to permanently archive cited web pages. *Journal of Medical Internet Research*, 7(5):e60, December 2005.

[16] Updated: Cligs Got Hacked - Restoration from Backup Started. http://j.mp/zg4Ft, June 2009.

[17] Hack update: Backup saves 93% of hacked urls. http://blog.cli.gs/news/hack-update, June 2009.