dead.drop: URL-based Stealthy Messaging

Georgios Kontaxis*, Iasonas Polakis†, Michalis Polychronakis* and Evangelos P. Markatos†
*Computer Science Department,
Columbia University, USA
{kontaxis, mikepo}@cs.columbia.edu
†Institute of Computer Science,
Foundation for Research and Technology – Hellas, Greece
{polakis, markatos}@ics.forth.gr

***Abstract*—In this paper we propose the use of URLs as a covert channel to relay information between two or more parties. We render our technique practical, in terms of bandwidth, by employing URL-shortening services to form URL chains of hidden information. We discuss the security aspects of this technique and present proof-of-concept implementation details along with measurements that prove the feasibility of our approach.**

I. Introduction

The Internet's public character mandates the need for confidential communications in respect to users' right to privacy. Cryptography may not always suffice on its own, as a user can be forced to surrender the key upon discovery of an encrypted message [1]. For that matter, covert channels offer the ability to communicate information in a clandestine way so as to avoid raising any suspicions.

At the same time, more than 50% of the Internet's traffic is attributed to the World Wide Web [1], which presents an excellent environment for piggybacking covert messages. Empirical observations suggest that URLs are getting longer and more complicated, increasing in depth and the number of parameters they carry. For instance, links to pages of Google Maps and Reader or Amazon may be 500 or 1000 characters long. As a result, one could hide meaningful information in such "noise" to establish a covert communication channel with another party. However, as the channel has to blend in, it is bound by the statistical properties of the URL population in the network.

URL shortening has evolved into one of the main practices for the easy dissemination and sharing of URLs. URL-shortening services provide their users with a smaller equivalent of any provided long URL. Currently, a plethora of such services exists [2] and are used extensively in a wide range of sites, including blogs, forums and social networks. Their popularity is increasing rapidly by as much as 10% per month, according to Alexa [3].

[1] http://www.schneier.com/blog/archives/2008/10/rubber_hose_cry.html

[2] http://www.prlog.org/10879994-just-how-many-url-shorteners-are-there-anyway.html

[3] http://www.alexa.com/siteinfo/bit.ly#trafficstats

In this paper we propose the use of URLs as a covert channel to communicate information between two or more parties. The basic idea is to encode the message using the URL-permitted subset of the US-ASCII character set [2] and masquerade it to resemble an actual URL. Such straightforward approach is limited in terms of the amount of data that can be encoded in each URL, due to the need of blending in with the population of actual URLs. We overcome such restrictions by employing URL-shortening services to form URL chains of hidden information. Multiple long, information-carrying, URL look-alikes are arranged and digested in reverse order and each resulting short URL is appended to the next long URL before it is shortened. In the end, each short URL unwraps into an information-carrying long URL which contains the pointer to the next short URL, thereby forming a chained sequence of messages. We discuss the security of our approach and present the proof-of-concept implementation details along with measurements that prove the feasibility of our approach.

The contributions of our work are the following:

- We make it practical, in terms of bandwidth, to employ URLs as a covert channel for communicating information between two or more parties.
- We propose a novel use of URL-shortening services for storing and transmitting short messages in a stealthy way.
- We have a prototype implementation of *dead.drop* as well as a sample application that uses our API to handle short files and message strings.
- We perform a security analysis of our proposed method and prove that it holds the following properties: it is unobservable against both static and dynamic analysis and robust against straightforward message extraction.
- We study the traits and features of the most popular URL-shortening services, along with the characteristics of short URLs from a popular social networking site, in respect to our goal of an unobservable communication channel.

The rest of the paper is organized as follows: in Section II we provide background on HTTP URLs and URL shortening services. In Section III we present our design for a covert messaging system. In Section IV we measure the feasibility of our proposal in relation to the traits and features of popular URL-shortening services and study a sample popular of short URLs from a popular social networking site. We present related work in Section VI and conclude in Section VII.

II. Background

In this section we provide some background on Uniform Resource Locators (URLs) and URL-shortening services. In our proposal, we employ URLs to carry messages in a covert fashion and utilize URL-shortening services for overcoming usual bounds and making the communication channel more feasible and efficient.

Uniform Resource Locators. URLs are used to refer to resources in protocols such as HTTP. An HTTP URL is comprised of the host part, identifying the network host serving content (e.g., web server), an absolute path to the resource requested (separated by slashes) and finally the query part, which begins with the questionmark character and may contain data to be transmitted towards the network host. Such data is split into variables separated by the ampersand (&) character. Furthermore, HTML pages may contain bookmarks to their content for easier user access. Appending the hash (#) character and the name of a bookmark at the end of an HTTP URL is acceptable so that the browser will navigate to the specified bookmark once the HTML content is loaded. Overall, a typical HTTP URL resembles the format http://host/subfolder/p.html?var1=val1&var2=val2#chapter2. URLs are written in a subset of the US-ASCII character set: alphabet letters, digits, the character plus ("+"), period (".") and hyphen ("-") are allowed. The rest of the ASCII characters are either part of the control set or unsafe and must be encoded in a hexadecimal representation. While originally only a few characters needed to be written in HEX, today it is common practice for URLs to contain non-Latin characters, e.g. to use words in Greek or Japanese. Such strings are written using the Unicode Transformation Format (UTF [3]) and then represented in hexadecimal form as the respective byte values do not fall within the acceptable character set. As a result, URLs like http://example.gr/%CE%B1%CE%B2%CE%B3/%CE%B4%CE%B5%CE%B6.html are not only common but may dominate network traffic at a local scale.

According to the RFC [4] for the HTTP protocol, there is no a priori limit for a URL's length. Microsoft [4] has placed a limit of 2,048 characters on the Internet Explorer browser. Empirical data on browsers like Mozilla Firefox and Google Chrome suggest that they can handle URLs that span to tens or even hundreds of thousands of characters. Furthermore, there is no specified limit on the depth of the URL (i.e. length of the path segment) or the number of variables or their maximum length. Such properties are implementation-specific and depend on the file system, web server instance and application running each time on the content host.

As mentioned earlier, URLs are resource locators, meaning that one describes a resource on a host computer and that resource may or may not exist. For instance, http://www.example.com/page1.html may exist and return an HTML page and http://www.example.com/page2.html may not and yet both URLs are considered valid in syntax. Therefore, the only way to evaluate a URL is by trying to access the resource it describes. Even then, a failure is acceptable as it is common for resources to be relocated or mistyped (e.g. user makes a spelling error while typing a URL in the browser). Furthermore, what happens when a requested resource is not available depends upon the implementation of the specific web server and application. For instance, the web server may respond with a default HTML page (e.g. the contents of the home page) or a page with an application-specific error string. It may also respond with an HTTP error 404 "Not Found" or with an HTTP 301 redirection message to a default valid resource (e.g. URL of the home page). Therefore, it is very difficult to determine an actual URL from a look-alike URL as access requests may fail silently. Even if they fail, such behavior is not definitive for characterizing a URL-like scheme. Even if there are a few cases that a fake URL path returns a standard type of error, when it comes to URL parameters (or query variables), server behavior is ever more relaxed: invalid variable values may be simply ignored or return an application-specific error string. Unknown variables will always be ignored. Finally, while hash locators (#) are never transmitted to the server as part of a request, it is acceptable for URLs to carry one of unspecified length as they having meaning inside the web browser.

Overall, we consider it very hard to distinguish a look-alike URL (i.e. encoded information masqueraded to follow the same structure) from an actual URL. Such identification is hard by statically examining the two strings, such as checking for a similar length and number of parameters. Also, as mentioned earlier, it is hard to dynamically distinguish them by attempting to access the resource they are supposed to refer to. In light of that, we propose the use of messages encoded in look-alike URLs and their subsequent transmission or storage in a shared space, to form a covert communication channel.

In order for look-alike URLs to be indistinguishable from actual URLs in static analysis, they must carry the same statistical properties in terms of structure (i.e. length, depth and parameters). One could argue that maintaining these properties severely limits the bandwidth of the communication channel. For that matter, we employ URL-shortening

[4] http://support.microsoft.com/kb/q208427/

services to implement URL chains. We will now discuss such services and present the concept of URL chains in section III.

URL-shortening Services. URL shortening services provide a persistent mapping from a long URL to a short one. Their contribution resides in the fact that they provide an efficient way for users to share long HTTP addresses. Instead of posting the original URL, a short address that redirects to the original one is supplied by the service and can be posted in web pages, blogs, forums, social networking sites, e-mails and instant messengers that are either incapable of handling long inputs or inefficient in presenting them. For example, if the user submits http://www.this.is.a.long.url.com/indeed.html to bit.ly, a prominent shortening service, it will return the following short URL: http://bit.ly/dv82ka. Any future access to http://bit.ly/dv82ka will be redirected, by bit.ly, to the original URL through an "HTTP 301 Moved Permanently" response [4]. For any given input, the returned short URL is comprised of an identifier 5-7 characters long under the domain of the service. Some of these services return a random short URL while others also support customized ones. For example, the tinyURL service allows customized short URLs, such as http://www.tinyurl.com/customalias. Furthermore, such mappings are kept indefinitely in the vast majority of services, if not all of them.

Different limitations are posed by each service regarding the length and form of URLs to be shortened and, in some cases, input several megabytes long is accepted. Furthermore, certain services do not check whether the submitted content has a valid URL format, meaning it does not even have to start with http:// while all special HTML characters, from white spaces to symbols, are accepted and correctly handled. This behavior can be exploited to hide arbitrary content in URL shorteners. In section IV we present the traits and features of the most popular URL-shortening services.

Despite the different limitations, the common characteristic of all these services is that they store information (a long URL) and point to it via a pointer (the shortened version of the URL). Additionally, if the URL-shortening service is known, one may only refer to a short URL by its identifier which is only 5-7 characters long, resulting in a very efficient way to address a very long string of information using a pointer of less than 10 characters.

III. DEAD.DROP

A. Design and Operation

The idea behind *dead.drop* is that messages are masqueraded as one or more look-alike HTTP URLs and subsequently shortened by one or more URL-shortening services. The shortening process conceals the true length of the information-carrying URL and makes it easier to blend in a population of other URLs in the network, a social networking site or an instant messaging session. Furthermore, URL-shortening exhibits some interesting characteristics that strengthen the covert nature of the system and support advanced features such as message chains, which are explained later on.

```
http://maps.google.com/?q=%E7%A5%9E%E5...
http://maps.google.com/?q=%56%47%68%70...
```

Listing 1. A real URL with unicode characters encoded in HEX versus a look-alike URL carrying a hidden message.

Message Encoding. Message strings are encoded into a Base64 chunk and that chunk is split into blocks of arbitrary size. These blocks are masqueraded as a look-alike HTTP URL, either as part of its path or values to query variables or location hashes.

For instance, the message string "This is a secret." will become "VGhpcyBpcyBhIHNlY3JldC4=" and that Base64 chunk will be split into blocks and form a URL like the following: http://cnn.com/VGhpcy/html/BpcyBhIH/en/news?id=NlY3#JldC4%%3D. That URL will be subsequently shortened using a URL-shortening service, such as bit.ly to something like http://bit.ly/fGvq5h and the final short URL will be the token which will be kept as a pointer to the original message, or transmitted across communicating parties.

One could argue that the forged URL stands out, as apparently meaningless strings of characters are present in its path. However, we consider it hard to distinguish the fake URL in the example of Listing 1 where the look-alike URL is masqueraded as if carrying characters in the Unicode set which are being represented by their hexadecimal values. On a side note, the real URL is 205 characters long while the message-carrying one is less than half its size.

The length of each block and their dissemination in the various segments of the URL is deterministic, so that the recipient of the hidden message may reverse it. Nevertheless, it is based upon a pre-shared, secret, random seed between the sender and receiver. Moreover, using that seed, blocks are not placed in order in the URL but are transpositioned. In the case of multiple URLs, such a transposition takes place between different URLs so that two sequential blocks of the original message are located in unrelated URLs. The seed is essentially the "dictionary" that such protocols require so that special meaning is assigned to seemingly meaningless things. Furthermore, the secret determines which URL-shortening services will be used and in which order. The same thing happens with the selection of the domains in the prefix of the URL (e.g. cnn.com), which are selected from a pool based on the value of the secret. The secrecy of the random seed makes it harder for the adversary to distinguish a look-alike URL from an actual URL by statically observing their structural properties. However, even if the secret is revealed, we consider it non-trivial for one to detect an information-carrying URL. The secrecy of the seed makes it also hard to recover the hidden message from a captured and

```
 [IN STR] DATA1DATA2DATA3
 [STEP 1] http://cnn.com/#DATA3# ->
          http://bit.ly/e2V3N0
 [STEP 2] http://cnn.com/#DATA2#
          http://cnn.com/#DATA2#e2V3N0 ->
          http://bit.ly/evkkYf
 [STEP 3] http://cnn.com/#DATA1#
          http://cnn.com/#DATA1#evkkYf ->
[OUT URL] http://bit.ly/gGPWL1
```

Listing 2. Example creation of a URL chain.

known-to-be information-carrying URL. We do not consider our encoding to be strong against cryptanalysis techniques but the original message can be encrypted prior to its hiding to preserve its secrecy even upon recovery.

On can argue that as the amount of encoded information increases, the URL's length will make it stand out among others in a network trace. For that matter we measure the amount of long URLs from a bit.ly trace we have obtained from the Twitter social network. It turns out that 10% is longer than 200 characters and 1% of the set is about 2K characters long. We consider a range of 200-2000 characters for encoding information in a single URL. In section IV we present our findings in a more analytical fashion.

Short-URL Chains. To share messages longer than 2K characters we use more than one information-carrying URL and in each one append pointers towards the next. Such practice is made possible by the use of URL-shortening services that digest the long URLs and return very short pointers. In detail: the message is encoded into multiple 2K-character Base64 chunks. Each chunk is split into blocks and encoded in its own look-alike long URL. Each URL is shortened, starting from the last one moving towards the first one. Listing 2 provides an example for creating such chain of URLs. So if we want to share 3 2K-character chunks, we will shorten the URL of chunk 3, next the URL of chunk 2 and finally the URL of chunk 1. The short URL produced by URL 3 is appended to the end of URL 2, prior to its shortening (steps 1 and 2). And the short URL produced by URL 2 is appended to the end of URL 1, prior to its shortening (step 3). Communicating party 'A' will only have to transmit the final short URL (http://bit.ly/gGPWL1) to party 'B'. When 'B' tries to unwrap the short URL, he will extract the encoded information and find the address of the next short URL appended at the end.

B. Implementation

We have implemented *dead.drop* using a high-level messaging and a low-level communication layer, similar to the OSI model. The message layer provides read/write functionality to applications using *dead.drop*. It is responsible for translating a *write()* function call to a series of message-splitting-into-blocks operations, masquerading the information into URLs and, finally, digesting them into short URLs. It is also responsible for translating a *read()* function call into a short URL, decoding the URLs and, finally, reassembling the message. The communication layer is aware of URL shortening services and interacts directly with them. It implements the logic of *dead.drop* at the lowest layer using the URL-shortening services' HTTP API to communicate with them for URL shortening/unshortening operations.

Our prototype implementation of *dead.drop* is written in the C programming language. We employ the libcurl library for handling the HTTP communication with the URL shortening services. We also incorporate functionality from the openSSL framework to perform AES encryption and SHA1 hashing and base64 encoding. We have exposed a C API, as part of the messaging layer, and written a sample application, titled shortFS, for uploading/downloading small files or message strings using the URL shortening services.

C. Use Case

Figure 1 presents an example interaction between two parties who wish to exchange messages in a stealthy way, using our system; Alice wishes to communicate message m to Bob. She splits the message into variable-size blocks and constructs look-alike URLs in reverse order, starting from the last block (step 2a). Each URL is shortened and appended to the next one (step 2b). Finally, the last URL is uploaded in Twitter. Considering that 25% of messages in this popular micro-blogging service contain some URL [5], Alice's action does not raise any suspicions. Bob receives the URL through Twitter, reverses the process (steps 4a, 4b) and reassembles Alice's message. As shown, real-time communication can be accomplished through a "live" channel such as a social network or blog.

IV. Study of URLs and short URLs

In this section we present several characteristics of the top URL shortening services that we leverage so as to incorporate them in *dead.drop*.

A. Profile of Shortening Services

Table I outlines the traits and features of the most popular URL shortening services, sorted by their Alexa [5] rank. Among their characteristics, we highlight whether they allow the user to *a priori* determine the returned short URL, their maximum allowed input URL length, the presence of a URL-format validation mechanism and, finally, whether there is an expiration date for their mappings.

The ability to customize the redirection URL enables us to address blocks by their hash, thus providing a more flexible design. Additionally, by selecting customized mappings from the namespace of a hash function we can avoid collisions with short URLs already registered with the service. The length of the input URL determines the block size we are able to upload, with larger sizes enabling more

[5] http://www.alexa.com/

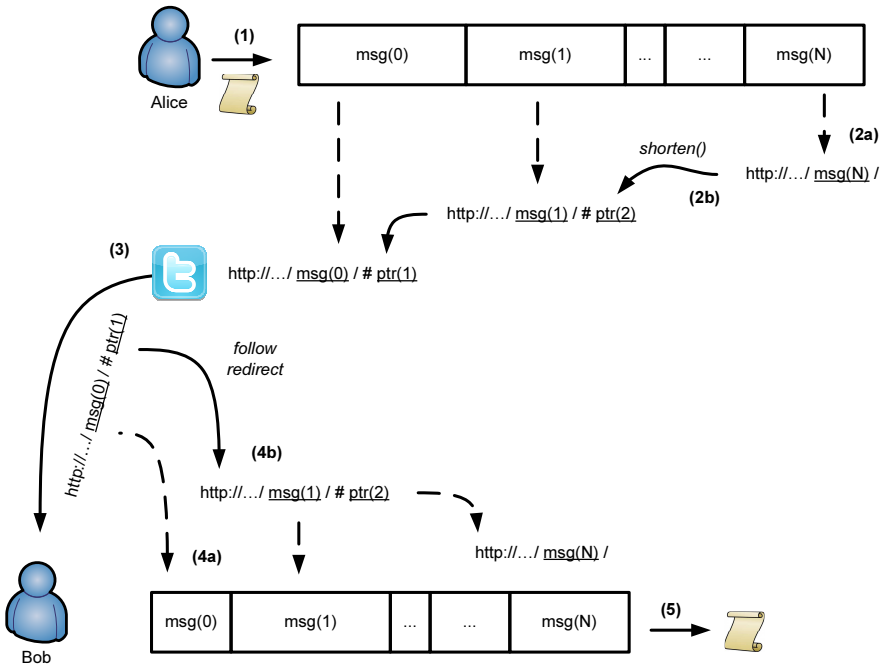

Figure 1. Use case for *dead.drop*; Alice splits message into variable-size blocks and constructs look-alike URLs in reverse order, starting from the last block (step 2a). Each URL is shortened and appended to the next one (step 2b). Bob receives a single URL, the one for message block zero, and recovers the message by reversing the process (steps 4a, 4b).

Service Name	Mapping Type	Input Max Chars	Input Check	URL Expir.
bit.ly	Custom	2,000	No	Never
ow.ly	Random	993	No	-
tinyurl	Custom	1,000,000	No	Never
su.pr	Custom	993	Yes	-
tiny.cc	Custom	8,000	Yes	Never
is.gd	Random	2,000	Yes	Never
cli.gs	Custom	1,000	Yes	-
xrl.us	Random	8,000	Yes	5 years
snipurl	Custom	8,000,000	No	Never
kl.am	Custom	65,528	No	-

Table I
TRAITS AND FEATURES OF THE 10 MOST POPULAR URL SHORTENING SERVICES.

efficient transfers. As seen in Table I, in the worst case, two URL shortening services allow us to upload 993 bytes, while in the case of snipurl we can upload blocks of over 7.5 million characters. Finally, five of the top ten services apply some form of input checking to make sure it complies with the URL format. However, as our goal is to blend in the population of URLs on the network, our *dead.drop* URLs are able to meet such demands.

B. Population of short URLs

Here we measure the properties of a large set of long URLs collected from the Twitter social networking site. We captured 1 million bit.ly short URLs and expanded them to acquire our dataset. In detail, we measure: the length of the long URLs, their depth (i.e. how many times the '/' character is encountered in the URL) and the number of parameters each one carries at the end.

As seen in Figure 2, 10% of the URLs is longer than 150 characters and 1% may be up to 2000 characters. This observation affects the amount of information that may be encoded in a single URL, as our goal is to blend in with the rest of the URLs on the network. However, the use of URL chains, as detailed in section III, allows us to transmit a much larger amount of information.

Figure 3 gives an idea of the depth of the long URLs, i.e. the number of levels, beneath the root of the web domain. We can see that 10% of the URLs has a depth of at least 7, meaning we can place at least 7 blocks of information as part of the URL's path, as demonstrated in section III.

The same figure, also depicts the number of parameters each long URL has. One may notice that 10% of the URLs has more than 3 parameters. As a result, we can place more than 3 blocks of information, masqueraded as URL parameters, and have the URL blend among others.

To sum up, our measurements show that we can place hidden information in a legitimate-looking URL and blend in with 10% of the URLs in our set in the following way: we can include at least 7 blocks of information as part of the URL's path and at least 3 blocks of information as part a URL parameter. At any time, the size of each of those blocks must be such that the total length of the URL remains

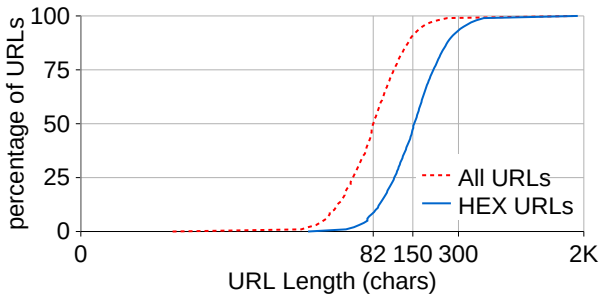

Figure 2. Length of 1M sampled URLs.

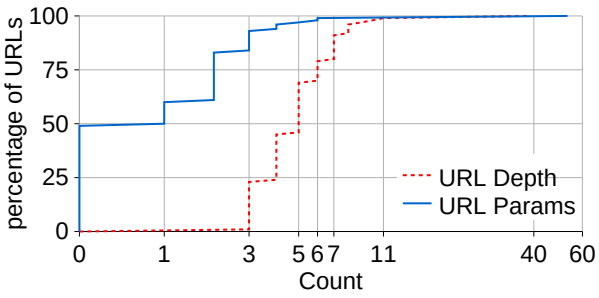

Figure 3. URL Depth and number of parameters.

between 150 and 2000 characters.

Evidently, there is obviously a tradeoff between unobservability and channel capacity; a tradeoff present in all covert channels. One could blend in with more than 10% (e.g. more than 30%) of a URL population by decreasing the block size of his messages to less than 150 characters (e.g. 100 characters).

C. Failure Transparency

In section II we talked about how Web servers handle requests for invalid resources transparently and therefore it is not always possible for someone to determine if a URL is valid or not by simply inspecting the HTTP response from the Web server. To verify this behavior we used the same population of 1 million short URLs, expanded them to their long equivalents and measured the extend to which failures in the corresponding Web sites were transparent. In detail, for each URL we tried accessing it, accessing the root of the domain (e.g. http://www.example.com if the URL was http://www.example.com/test/page.html) and accessing a random path under the domain which was certain not to exist. Overall, in 20% of the cases the Web servers returned the same HTTP response code (HTTP 200) for both the valid URL and the invalid URL (random path under valid domain).

V. Security Analysis

A. Threat Model

Our adversary is a person with monitoring access to the network, e.g. an administrator. He may be close to the message sender or receiver or somewhere in the network path between them. He is able to conduct deep packet inspection, extract packet segments, e.g. HTTP headers, and perform string matching. He routinely monitors network traffic of instant messaging applications, e-mail and communication with online forums and social networks (e.g. Twitter). He is aware of our technique and tries to a) identify look-alike URLs and b) recover their hidden message.

Our security requirements are a) to be very hard for the adversary to distinguish a look-alike URL among a population of URLs on the network and b) to require some advanced form of analysis (e.g. cryptanalysis) in order to extract the hidden message.

We also require that it is very hard for the adversary to proactively apply some form of transformation on the network packets as they leave or enter his network, so that, without knowledge of a specific look-alike URL he is able to destroy the hidden message while preserving the functionality of a real URL, i.e., allowing a user to access the actual resource being referenced.

B. Unobservability

Static Analysis. Such analysis tries to identify a look-alike URL string in a population of URL strings. The statistical and structural properties of URLs are employed to highlight anomalies in the format, length, depth and number of parameters of the URL. Additionally, semantical heuristics can be employed to highlight URLs that appear non-readable by humans. Our goal in *dead.drop* is to conform to the statistical and structural properties of a population of URLs. In section IV we discussed how we are able create look-alike URLs with the same properties as 10% of a sample population of URLs. In addition, as URL-shortening services are already very popular, their use by *dead.drop* and their presence in any communication cannot constitute a heuristic by which look-alike ULRs are identified. What is more, as *dead.drop* employs a plethora of shortening services, as well as a great variety of real domains that are included in the look-alike URLs, the covert channel cannot be characterized by the persistency of certain services or domain strings. Furthermore, while URLs used to be human-readable, the common practice of including the hexadecimal values of characters from UTF-8 charset to support URLs in non-English languages has diminished that expectation. As demonstrated in section III look-alike URLs can assume such forms so that they are indistinguishable from actual URLs. Moreover, in the sample population of 1M URLs studied in section IV, more than 5% of them belong to this category and we expect such a percentage to be much higher or even be dominant in more localized URL samples in countries like Greece or Japan.

Dynamic Analysis. Such analysis tries to evaluate a URL

and decide upon its validity. The adversary tries to access the resource referenced by the URL and decide based on the web server's response whether it is an actual URL or a look-alike. As mentioned in section II the web server's or application's response to an invalid request is implementation-specific, may silently fail and even if it fails hard, such behavior it not definitive for characterizing a URL-like scheme as the web is already populated with broken URLs [6].

C. Robustness

Assuming an adversary has captured a known-to-be information-carrying URL, our goal is to render the use of some advanced form of analysis (e.g. cryptanalysis) mandatory for the extraction of the hidden message. The purpose of *dead.drop* is not to preserve the secrecy of the message once its container, the URL, has been discovered. For that matter, the message can be encrypted using some symmetric or asymmetric cryptographic function and then covertly transmitted using *dead.drop*. Nonetheless, message extraction is not as trivial as concatenating the parts of a URL string.

As described in section III, the way the message is encoded and hidden is based upon a pre-shared, secret, random seed between the sender and receiver so that two sequential blocks of the original message are located in unrelated segments of a URL or multiple URLs.

URL chains pose a certain risk, because if an adversary discovers one link, he may be able to recover a subset or all the URLs in that chain by following the address of the next short URL. For that matter, the parts carrying the encoded information from all the previous URLs are XOR-ed with each other to form a keystream and the first k-characters of that keystream are used to XOR the k-character address of the next short URL. That way, one is unable to read the actual address of the next short URL, from the end of the current URL, unless he is in possession of all previously exchanged URLs.

D. Countermeasures by the Adversary

As mentioned earlier, one of the requirements for *dead.drop* is to be very hard for the adversary to proactively apply some form of transformation to all URLs in packets, entering or leaving his network, that would destroy hidden information in look-alike URLs while leaving actual URLs intact. We consider such practice to be infeasible at the network level and discuss the case of a transparent HTTP proxy at the application level.

An adversary close to the receiver can install a transparent HTTP proxy so that outgoing HTTP requests are intercepted and handled by the proxy. Initially, the receiver places a request towards a URL-shortening service that is expected to return a long look-alike URL. The URL-shortening service responds with an HTTP 301 redirection message to the request, containing the long URL. In the case of a transparent proxy, the receiver is the proxy's client. If the proxy places the request on behalf of the client, there are two possible scenarios: follow-up on the redirection and serve the final response, i.e. HTML page, to the client or return the redirection header and let the client issue a new request towards the domain of the long URL.

If the proxy does the latter, the client immediately becomes aware of the long look-alike URL and is able to recover the hidden message. If the proxy is configured to silently follow-up on the redirection, the client is deprived access to the look-alike URL string, which contains the hidden information, and is provided with only the outcome of the redirection. In a redirection towards an actual URL, the right HTML page will be returned so that legitimate URLs are not affected. However, in the case of a look-alike URL the important part is the URL itself and not the response, which may not even exist. This is a traffic transformation case where legitimate URLs are not affected but look-alikes are destroyed. However, this type of configuration is considered impractical and insecure as it allows a third-party to poison the proxy's cache or, if a zero-cache proxy is used, bypass the same-origin-policy security mechanism in the clients' web browser. For that matter we find it unlikely for such a solution to be widely adopted. Nevertheless, if such practice is present, in *dead.drop* we can hide messages in look-alike URLs using the HTTPS scheme which prompts a TLS session with the remote host. Proxies are not able to relay an encrypted session and as a result the long URL must be returned to the client.

E. Countermeasures by shortening services

As we employ URL-shortening services to form URL chains or to achieve first-level obscurity regarding the contents of a look-alike URL, we consider possible measures that can be taken by these services to prevent us from utilizing them. It is our belief that the type of use taking place does not deviate from standard usage nor does it violate their terms of service.

Rate limiting. By limiting the amount of requests a user can issue in a specific time interval, a shortening service can affect a user transmitting a large volume of hidden messages with the use of URL chains. However, *dead.drop* aims at using many shortening services in parallel to avoid being characterized by the use of a specific service. As a result, the number of URL chains created is distributed among many different services. Furthermore, several news sites or blogs routinely utilize such services to acquire short links for a large volume of articles as they are published.

Reduction of acceptable URL length. *dead.drop* traffic must blend-in with actual URLs, and for that matter their statistical properties such as URL length, must be preserved. As a result, if any kind of input-length reduction is applied, it is expected to affect a portion of the legitimate population of

[6] http://try.powermapper.com/demo/statslinks.aspx

URLs, something not desired by a URL-shortening service. In section IV we discussed how we blend-in with 10% of the legitimate URLs, a significant portion that we are sure is not to be discarded by these services. Furthermore, we present that the vast majority of services permits input many times longer that the bounds we have set for *dead.drop*. Finally, let it be noted that the HTTP RFC does not define a limit to the length of a URL.

Content inspection. Heuristics can be employed to verify that submitted content follows the format of URLs, so as to prevent the submission of arbitrary content. In section IV we showed that only half of the top URL-shortening services conduct such a check. However, as one of the requirements of *dead.drop* is to masquerade the hidden information according to the structure of a legitimate URL, we are not affected by such an inspection. Furthermore, some shortening services attempt to verify the existence of the domain or the actual page the long URL points to. Even if they decide to do so, the ambiguity of web applications when handling erroneous input, as we have already discussed in section III, allows us to overcome such measures.

URL mapping expiration. In section IV we showed that 90% of the top URL-shortening services preserves mappings indefinitely. Even if such practice ceases, the goal of a covert communication channel, such as *dead.drop*, is to allow the transmission of short-lived messages and therefore mapping expiration will not affect us.

VI. Related Work

Traditionally covert channels aim at storing information that has to remain concealed in deprecated or unused spaces inside network protocol headers.

In [6] the authors overload flags and fragmentation fields in the IPv4 packet header to communicate “1” and “0” bits between two cooperating parties. In [7] bit signaling between two parties is achieved by utilizing the DNS query recursion flag. To signal a bit of “1”, the sender issues a DNS request with the recursion flag set. The receiver then issues a query for the same hostname with the recursion flag off. If he receives a valid record, then the sender is signaling a bit of “1”. If no record is returned, the sender is signaling a bit of “0”. In [8] the authors construct message exchange through mixnets using a set of cooperating web servers which include non-involved subjects who provide cover traffic and propagate the messages for their protocol. They masquerade the message reads and writes as a series of HTTP requests towards CGI scripts. Oblivious web surfers visit a series of carefully crafted pages, execute embedded Javascipt code in their web browsers and, by that, facilitate the transfer of messages among the mixnet nodes. Hiding information in images [9] is perhaps one of the most popular and well-know techniques. A common practice is to overload the LSB with the bits to be signaled. Variations in the timing of events, for instance in a network, may be employed to signal information between two parties [10].

Although techniques as the ones aforementioned may be hard to detect, it is quite easy to take preemptive actions against them so as to neutralize and prevent potential message recipients from accessing the information. For instance, one could clear IP flags and fields that are not used in a network, delay or reorder the arrival of events (e.g. in network packets) or apply some form of modification to the image (e.g. compression) that will not alter the visual outcome but destroy any structure-specific information. Furthermore, in the case of mixnets, a supporting infrastructure is required, something which significantly impacts the practicality of this technique.

Our proposed method of encoding information, in URLs that blend-in, aims at addressing such weaknesses: our goal is to make the detection of a URL, carrying encoded information, hard and at the same time impossible for the adversary to apply any transformation to the URL that will destroy any encoded information but still remain functional.

Nikiforakis et al. [11] assess the security features of file hosting services (FHS). Such services return a secret URI for each uploaded file, which users can then divulge to other people. In this sense, such services are similar to URL shortening services, as they return a pointer to a specific resource. While users expect that their files will only be accessed by people they have revealed the URI to, experiments demonstrated that for several services adversaries can predict how secret URIs are create and access private files.

Antoniades et al. [12] conduct an extensive study regarding the characteristics and access patterns of short URLs. ++++

VII. Conclusion

In this paper, we present a technique for covertly relaying information by hiding it in URLs. Our system forges URLs that contain the information in various segments, such as values of variables or subfolders located on the remote web server. These URLs are subsequently digested through URL shortening services, and in the case of multiple URLs, we create chains of short URLs to hide the information. As an important goal of our system is to create URLs that cannot be distinguished from others in the network, we conducted a study concerning the characteristics of URLs such as their length, depth and number of variables. Based on those, we craft URLs with characteristics found in 10% of the URLs in the network. We also performed a security analysis of our proposed method.

Acknowledgments

This work was supported in part by the FP7-PEOPLE-2009-IOF project MALCODE funded by the European Commission under Grant Agreement No. 254116. Iasonas Polakis and Evangelos Markatos are also with the University of

Crete. Most of the work of Georgios Kontaxis was done while at FORTH-ICS.

REFERENCES

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*.

[2] "Rfc 1738 - uniform resource locators (url)," http://www.rfc-editor.org/rfc/rfc1738.txt.

[3] "Rfc 2279 - utf8, a transformation format," http://www.ietf.org/rfc/rfc2279.txt.

[4] "Rfc 2616 - hypertext transfer protocol – http/1.1," http://www.rfc-editor.org/rfc/rfc2616.txt.

[5] "TechCrunch - Twitter Seeing 90 Million Tweets Per Day, 25 Percent Contain Links," http://techcrunch.com/2010/09/14/twitter-seeing-90-million-tweets-per-day/.

[6] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. Workshop on Multimedia Security at ACM Multimedia '02*, December 2002.

[7] "The dns dead drop," http://landonf.bikemonkey.org/code/security/DNS_Dead_Drop.20060128201048.26517.luxo.html.

[8] M. Bauer, "New covert channels in http: adding unwitting web browsers to anonymity sets," in *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, ser. WPES '03, 2003.

[9] R. Ch, M. Kharrazi, and N. Memon, "Image steganography and steganalysis: Concepts and practice."

[10] I. S. Moskowitz, I. S. Moskowitz, A. R. Miller, and A. R. Miller, "Simple timing channels," in *Proceedings 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, 1994.

[11] N. Nikiforakis, M. Balduzzi, S. Van Acker, W. Joosen, and D. Balzarotti, "Exposing the lack of privacy in file hosting services," in *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, ser. LEET'11.

[12] D. Antoniades, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. P. Markatos, and T. Karagiannis, "we.b: the web of short urls," in *Proceedings of the 20th international conference on World wide web*, ser. WWW '11. ACM.