Are you reading this? Feel free to edit and comment.
Please insert -> comment to add comments rather than adding them to the actual text.

See also discussion: https://plus.google.com/u/0/103112149634414554669/posts/FMDg2Vht8sb

Authors

Sai, Alex Fink

Overview

Currently, canonical Tor .onion URLs consist of a naked 80-bit hash[1]. This is not something that users can even recognize for validity, let alone produce directly. It is vulnerable to partial-match fuzzing attacks[2], where a would-be MITM attacker generates a very similar hash and uses various social engineering, wiki poisoning, or other methods to trick the user into visiting the spoof site.

This proposal gives an alternative method for displaying and entering .onion and other URLs, such that they will be easily remembered and generated by end users, and easily published by hidden service websites, without any dependency on a full domain name type system like e.g. namecoin[3]. This makes it easier to implement (requiring only a change in the proxy).

This proposal could equally be used for IPv4, IPv6, etc, if normal DNS is for some reason untrusted.

This is not a *petname* system[4], in that it does not allow service providers or users[5] to associate a name of their choosing to an address[6]. Rather, it is a *mnemonic* system that encodes the 80 bit .onion address into a meaningful[7] and memorable sentence. A full petname system (based

---

[1] https://trac.torproject.org/projects/tor/wiki/doc/HiddenServiceNames
https://gitweb.torproject.org/torspec.git/blob/HEAD:/address-spec.txt
[2] http://www.thc.org/papers/ffp.html
[3] http://dot-bit.org/Namecoin
[4] https://en.wikipedia.org/wiki/Zooko's_triangle
[5] https://addons.mozilla.org/en-US/firefox/addon/petname-tool/
[6] However, service operators *can* generate a large number of hidden service descriptors and check whether their hashes result in a desirable phrasal encoding (much like certain hidden services currently use brute force generated hashes to ensure their name is the prefix of their raw hash). This won't get you whatever phrase you want, but will at least improve the likelihood that it's something amusing and acceptable.
[7] "Meaningful" here inasmuch as e.g. "Barnaby thoughtfully mangles simplistic yellow camels" is an absurdist but meaningful sentence. Absurdness is a feature, not a bug; it decreases the probability of mistakes if the scenario described is not one that the user would try to fit into a template of things they have previously encountered IRL. See research into linguistic schema for further details.

on registration of some kind, and allowing for shorter, service-chosen URLs) can be implemented in parallel[8].

This system has the three properties of being secure, distributed, and human-meaningful — it just doesn't also have *choice* of name (except of course by brute force creation of multiple keys to see if one has an encoding the operator likes).

This is inspired by Jonathan Ackerman's "Four Little Words" proposal[9] for doing the same thing with IPv4 addresses. We just need to handle 80+ bits, not just 32 bits.

It is similar to Markus Jakobsson & Ruj Akavipat's FastWord system[10], except that it does not permit user choice of passphrase, does not know what URL a user will enter (vs verifying against a single stored password), and again has to encode significantly more data.

This is also similar to RFC1751[11], RFC2289[12], and multiple other fingerprint encoding systems[13] (e.g. PGPfone[14] using the PGP wordlist[15], and Arturo Filatsò's OnionURL[16]), but we aim to make something that's as easy as possible for users to remember — and significantly easier than just a list of words or pseudowords, which we consider only useful as an active confirmation tool, not as something that can be fully memorized and recalled, like a normal domain name.

Requirements

1. encodes at least 80 bits of random data (preferably more, eg for a checksum)
2. valid, visualizable English sentence — not just a series of words[17]
3. words are common enough that non-native speakers and bad spellers will have minimum difficulty remembering *and* producing (perhaps with some spellcheck help)
4. not syntactically confusable (e.g. order should not matter)

---

[8] https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/ideas/xxx-onion-nyms.txt
[9] http://blog.rabidgremlin.com/2010/11/28/4-little-words/
[10] http://fastword.me/

[11] https://tools.ietf.org/html/rfc1751
[12] http://tools.ietf.org/html/rfc2289
[13] https://github.com/singpolyma/mnemonicode
http://mysteryrobot.com
https://github.com/zacharyvoase/humanhash
[14] http://www.mathcs.duq.edu/~juola/papers.d/icslp96.pdf
[15] http://en.wikipedia.org/wiki/PGP_word_list
[16] https://github.com/hellais/Onion-url
https://github.com/hellais/Onion-url/blob/master/dev/mnemonic.py
[17] http://www.reddit.com/r/technology/comments/ecllk

5. short enough to be easily memorized and fully recalled at will, not just *recognized*
6. no dependency on an external service
7. dictionary size small enough to be reasonable for end users to download as part of the onion package
8. consistent across users (so that websites can e.g. reinforce their random hash's phrase with a clever drawing)
9. not create offensive sentences that service providers will reject
10. resistant against semantic fuzzing (e.g. by having uniqueness against WordNet synsets[18])

Possible implementations

1. Have a fixed number of template sentences, such as:
   - Adj subj adv vtrans adj obj
   - Subj and subj vtrans adj obj
   - … etc

For a 6 word sentence, with 8 (3b) templates, we need ~12b (4k word) dictionaries for each word category.

If multiple words of the same category are used, they must either play different grammatical roles (eg subj vs obj, or adj on a different item), be chosen from different dictionaries, or there needs to be an order-agnostic way to join them at the bit level. Preferably this should be avoided, just to prevent users forgetting the order.

2. As (1), but treat sentence generation as decoding a prefix code, and have a Huffman code for each word class. We suppose it's okay if the generated sentence has a few more words than it might, as long as they're common lean words. E.g., for adjectives, "good" might cost only six bits while "unfortunate" costs twelve.

Choice between different sentence syntaxes could be worked into the prefix code as well, and potentially done separately for each syntactic constituent.

3. *This method is flawed; the template code bits are unrecoverable.*
Treat the first *n* bits as designating a template code.

Each template specifies phrasal constituents, together with the required bitlength and binding material — e.g. *"The adj4 n11 adv10 vtrans12 n8 in loc9 with a(n) adj9 n11"* would designate an

[18] http://wordnet.princeton.edu/wordnet/man2.1/wnstats.7WN.html

interpretation of the following 4+11+10+12+8+9+9+11 = 74 bits. Each constituent is simply looked up in the appropriate frequency-sorted word-class dictionary, as (2), so e.g. adj4 is a very common adjective, whereas vtrans12 is a moderately rare transitive verb.

Bitwidths smaller than the dictionary are 0-padded on the left for lookup (i.e. there are only 2x more 5-bit words than 4-bit; they are not separate).

There can be a fairly large number of templates if the dictionary sizes across constituent classes is adequate; for instance, supposing that one uses 6 words of possible bitwidth 4...12, required to total 60 bits, there are 5853 (~12.5 bits)[19] possible permutations among the bitwidths. This would require a further ~7.5 bits (~181) further information in the header, which could be achieved using alternative basic templates as in (1).

Alternatively, if we have 7 words, we can use a much smaller 10 bit (~1k word) dictionary and still take up ~12.8 bits using this method. Etc.

Usage

To form mnemonic .onion URL, just join the words with dashes or underscores, stripping minimal words like 'a', 'the', 'and' etc., and append '.onion'. This can be readily distinguished from standard hash-style .onion URLs by form.

Translation should take place at the client — though hidden service servers should also be able to output the mnemonic form of hashes too, to assist website operators in publishing them (e.g. by posting an amusing drawing of the described situation on their website to reinforce the mnemonic).

After the translation stage of name resolution, everything proceeds as normal for an 80-bit hash onion URL.

The user should be notified of the mnemonic form of hash URL in some way, and have an easy way in the client UI to translate mnemonics to hashes and vice versa. For the purposes of browser URLs and the like though, the mnemonic should be treated on par with the hash; if the user enters a mnemonic URL they should not become redirected to the hash version. (If anything, the opposite may be true, so that users become used to seeing and verifying the mnemonic version of hash URLs, and gain the security benefits against partial-match fuzzing.)

Ideally, inputs that don't validly resolve should have a response page served by the proxy that uses a simple spell-check system to suggest alternate domain names that *are* valid hash

[19] https://plus.google.com/u/0/103112149634414554669/posts/DLfvB76Zhav

encodings. This could hypothetically be done inline in URL input, but would require changes on the browser (normally domain names aren't subject so spellcheck), and this avoids that implementation problem.

International support

It is not possible for this scheme to support non-English languages without
a) (usually) Unicode in domains (which is not yet well supported by browsers), and
b) fully customized dictionaries and phrase patterns per language

The scheme *must not* be used in an attempted 'translation' by simply replacing English words with glosses in the target language. Several of the necessary features would be completely mangled by this (e.g. other languages have different synonym, homonym, etc groupings, not to mention completely different grammar).

It is unlikely *a priori* that URLs constructed using a non-English dictionary/pattern setup would in any sense 'translate' semantically to English; more likely is that each language would have completely unrelated encodings for a given hash.

We intend to only make an English version at first, to avoid these issues during testing.