UTBox: URL Toolbox

UTBox is a set of building blocks for Splunk specially created for URL manipulation. UTBox has been created to be modular, easy to use and easy to deploy in any Splunk environments. One of the core feature of UTBox is to correctly parse URLs and complicated TLDs (Top Level Domain) using the Mozilla Suffix List. Other functions like shannon entropy, counting, suites, meaning ratio, bayesian analysis, etc, are also available.

UTBox has firstly be created for security analysts but may fit other needs as it's a set of building blocks. UTBox only needs to be deployed on Splunk Search Heads (the bundles will automatically be sent to your Splunk Indexers). Finally, each lookups is shipped with a macro to make it easier to use.

What is what ?

The syntax of a URL is as follow:

```
scheme://[user:password@]domain:port/path?query_string#fragment_id
```

Component details:

- The **scheme**, which in many cases is the name of a protocol (but not always), defines how the resource will be obtained. Examples include http, https, ftp, file and many others.
- The **domain name** or literal **numeric IP address** gives the destination location for the URL.
- The **port number**, given in decimal, is optional; if omitted, the default for the scheme is used (80 for http, 443 for https, etc).
- The **path** is used to specify and perhaps find the resource requested.
- The **query** string contains data to be passed to software running on the server. It may contain name/value pairs separated by ampersands, for example *?first_name=John&last_name=Doe*.
- The **fragment** identifier, if present, specifies a part or a position within the overall resource or document. When used with HTML, it usually specifies a section or location within the page, and used in combination with Anchor elements or the "id" attribute of an element, the browser is scrolled to display that part of the page.

Source: http://en.wikipedia.org/wiki/Uniform_resource_locator

Lookup & Macros

A generic lookup call in Splunk is of the format:

```
... | lookup <lookup_name> field AS <field>
```

For example:

```
... | lookup ut_parse_simple_lookup url AS cs_uri
```

UTBox also provides macros definition for each lookup to make it easier to call the lookups. In the previous example, the call would be:

```
... | `ut_parse_simple(cs_uri)`
```

It is important to understand that those macros are simply shortcuts to lookups call. One can use one or another depending on their tastes.

Lookups

- **ut__parse__simple(url)**

  Parse the given URL with python `urlparse()` function to extract the following fields: `ut_scheme`, `ut_netloc`, `ut_path`, `ut_params`, `ut_query` and `ut_fragment`. All the fields are set to `None` when they don't have a value.

  In case of failure, a log entry will be written to `$SPLUNK_HOME/var/log/splunk/utbox.log`

- **ut__parse(url, list)** or **ut__parse__extended(url, list)**

  Extended version of **ut__parse__simple()** which uses a list to extract the following fields: `ut_port`, `ut_domain`, `ut_tld`, `ut_domain_without_tld`, `ut_subdomain`, `ut_subdomain_count` and `ut_subdomain_parts`. All the fields are set to `None` when they don't have a value, except the `ut_port` which is set to `80` and `ut_subdomain_count` which is set to `0`. The field `ut_subdomain_parts` which contains json data is expanded with the Splunk `spath` command and fields like `ut_subdomain_level_1`, `ut_subdomain_level_2`, etc, are created to allow to manipulate the various parts of the subdomain. Note that the field `ut_subdomain_parts` is dropped by the macro because it creates `ut_subdomain_level_<id>` fields.

  The `list` parameter can be one of:

  - `mozilla` to use the Mozilla Suffix List (complete list which include entries like 'googlecode.com' as a TLD).

- `iana` to use the IANA Top Level Domain list (contains entries like 'UK' but will not contains 'CO.UK' for example).
- `custom` to use a 'custom' list
- `*` to use all of the 3 above lists (mozilla, iana and custom)

By default, the iana list is chosen and the custom list can be edited by adding/removing entries into `$SPLUNK_HOME/etc/apps/utbox/bin/suffix_list_custom.dat` (one TLD per line, lowecase, punycode format like xn-...).

The list parameter must be set with an eval command like in the following examples:

```
... | fields cs_url | eval list = "iana" | ‘ut_parse(cs_url, list)‘
... | fields cs_url | eval list = "custom" | lookup ut_parse_extended_lookup cs_url AS
```

In case of failure, a log entry will be written to `$SPLUNK_HOME/var/log/splunk/utbox.log`

- **ut__shannon(word)**

  Return the shannon entropy of the given word

- **ut__countset(word,set)**

  Return the number of occurrences of each characters in the provided word according to the provided set of characters. The set is a free form input but pre-defined presets are also available (see list below).

  ```
  ...  | eval set="0123456789abcdefABCDEF" | ‘ut_countset(word, set)‘
  ```

  The previous example define a set with lowercase and uppercase characters [a-fA-F] and digits. The same result can be achieve using the appropriate preset:

  ```
  ...  | eval set="@hexa_all@" | ‘ut_countset(word, set)‘
  ```

  List of existing presets (brackets are excluded of the sets):

  - `@digits@` : `[0-9]`
  - `@alpha@` : `[a-z]` - lowercase only.
  - `@alpha_up@` : uppercase only.
  - `@alpha_all@` : lowercase and uppercase.
  - `@alphanum@` : `[a-z0-9]` - lowercase only.
  - `@alphanum_up@` : uppercase only.
  - `@alphanum_all@` : lowercase and uppercase.
  - `@hexa@` : `[0123456789abcdef]` - lowercase only.
  - `@hexa_up@` : uppercase only.
  - `@hexa_all@` : lowercase and uppercase.

- `@base64@` : `@alphanum_all@` plus `[+/=]`
- `@vowels@` : `[aeiouy]` - lowercase only.
- `@vowels_up@` : uppercase only.
- `@vowels_all@`: lowercase and uppercase.
- `@consonants@` : `[bcdfghjklmnpqrstvwxz]` - lowercase only.
- `@consonants_up@` : uppercase only.
- `@consonants_all@` : lowercase and uppercase.
- `@noalpha@` : range 0x20 to 0x7F without `@alpha_all@`
- `@punct@` : range 0x20 to 0x7F without `@alphanum_all@`
- `@typo@` : `[&*@\^#%~_|]`
- `@word@` : only count the characters that actually are in the submitted word.

To ignore the case of the scanned words, use the `lower()` function from Splunk's `eval` command like in the following example:

```
... | eval set = "@word@" | eval field_lowered = lower(
field ) | lookup ut_countset_lookup word as field_lowered
set
```

Finally, each characters, encoded in its hexadecimal representation, is accessible after a call to the `spath` command like in the following example. A global `sum` counter is also available which is the total of counted chars:

`... | eval set = "@word@" | lookup ut_countset_lookup word as field set | spath input=ut_countset` (will give access to fields like `ut_countset.2E` for example)

For example, if you want to count the number of dots in the field `countme`, simply proceed this way:

```
... | eval countme="a.b.c.domain.tld" | eval set="." |
‘ut_countset(countme, set)‘| spath input=ut_countset | rename
ut_countset.sum AS "n_dots" | fields - ut_countset*
```

- In the previous example, we rely on the field `sum` because we only count for one specific char (the dot).
- To have the number of dots in a set larger than just the dot, we would have accessed to its individual counter represented by the value of 'dot' in hexadecimal ('.' = 0x2E). This could have been done in the previous example as well:

```
... | eval countme="a.b.c.domain.tld" | eval set="." | ‘ut_countset(countme, set)‘| spa
```

- **ut_suites(word, sets)**

  Return the longest suites of the submitted preset(s) (see `ut_countset()` above for a list of available presets). Some presets have no meaning when used with `ut_suites` like `@word@` for example. If you want to specify multiple presets, separate them using a comma like in the following example. Note that the macro drop the field ut_suites because it is automatically expanded with the `spath` command.

  Example: Get the maximum suites of vowels, digits and consonants:

  `... | eval set="@digits@,@vowels@,@consonants@" | ‘ut_suites(word, set)‘` This will ouput the new fields `ut_max_suite_digit`, `ut_max_suite_vowels` and `ut_max_suite_consonants`.

- **ut_meaning(word)**

  Return a ratio between the length of the word and all the words composing it based on a dictionnary (in the script directory). For example, the word `microsoft` will have a ratio of 1 because it's composed of the words `micro` and `soft` that are in the dictionnary. If the word `soft` would not have been in the dictionnary, the ratio would drop to 0.56 (len(micro) = 5, len(microsoft) = 9, 5/9 = 0.56). In short, this is usefull to identify DGA domains.

- **ut_bayesian(word)**

  Study the bigrams distribution of two input list of domain names to compute the probability that an input domain is either bad regarding the bigrams composing it. In short, this is usefull to identify DGA domains. The lists are in the bin directory of the app. The bad list come from malwaredomainlist.con and the good list comes from alexa.com. They both are around 27,000 entries.

- **ut_levenshtein(word1, word2)**

  return the levenshtein's distance between the two submitted words.