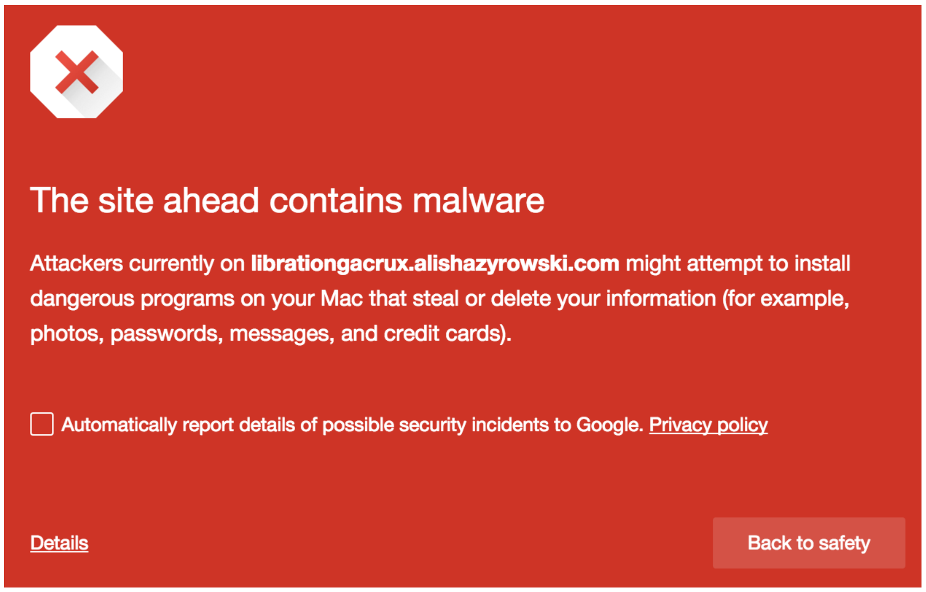
Google Safe Browsing API 3.0

Overview

The Google Safe Browsing App for Splunk allows checking URLs against Google's Safe Browsing service (a set of constantly-updated blacklists of suspected phishing, malware, and unwanted software pages). Through Google's API, the app will download an encrypted table for local, client-side lookups of URLs.

The current version of the Google API is 3.0 and this app has only been developed and tested regarding the specifications of the version 3.0 of the protocol.

Google Safe Browsing is heavily used by today's Web Browsers like Chrome as shown in the following screenshot:


The site ahead contains malware

Attackers currently on **librationgacrux.alishazyrowski.com** might attempt to install dangerous programs on your Mac that steal or delete your information (for example, photos, passwords, messages, and credit cards).

☐ Automatically report details of possible security incidents to Google. Privacy policy

Figure 1: Chrome alert - June 2015

Protocol explanation

Before installing this app, it's a good thing to be familiar with the basics of the protocol as well as some terms.

Google publishes phishing, malware, and unwanted software data in three separate blacklists (googpub-phish-shavar, goog-malware-shavar, and goog-unwanted-shavar). Each is a list of SHA-256 hash values that are usually truncated to a 4-byte hash prefix.

A hash prefix is the first bytes of a full hash, as shown in the following example:

- URL: malware.testing.google.test/testing/malware/
- '518640453f8b2a5f0d43bc225152f49530be2a40bfe2bab60aaaee7a67b10890' is the full length SHA-256 of the above URL
- '51864045' is the 4-byte hash prefix

This app will periodically interrogate Google Safe Browsing API to get, for the specified lists, the new chunks. A chunk contains 0 or more (1, 10, 100, 237, 542, 1,000, etc) hash prefixes. The chunks will be downloaded as soon as the protocol permit it and will be stored locally, in Splunk KVStore. On the other end, the Google Safe Browsing app provides lookups that can be used for real-time evaluation of your data against the local cache (KVStore).

There are two kinds of chunks:

- add chunks - matching entries in the list
- sub chunks - indicating false positives in the list

To see the magnitude size of these lists, in June 2015, the list goog-malware-shavar contained more than 67,000 chunks.

For a more in depth description of the protocol, please read the Google Specifications, as well as the initial specifications, still containing relevant information regarding the protocol.

Setup

1- Installing the App

First, install the app like any other Splunk app via the Web UI or the command line. This app is installed only on a Search Head.

Note that to update the lists, the Search Head requires Internet access to download the lists from Google. This app is not shipped with a copy of the databases of the hash prefixes.

2- Generate a Google API Key

Obtain an API key using a Google Account to utilize Google Safe Browsing service.

Please use the Google's developer's console and follow the **Getting Started** section of Google official documentation for instructions

This service is 100% free at the time of writing this documentation (June 2015).

3- Initial bootstrap

With the `api_key` available to access Google Safe Browsing service, proceed with the initial bootstrap script to quickly populate the local database and retrieve the chunks from the chosen lists.

The script `manual_bootstrap.py` has been designed to allow the initial blacklists retrieval from Google, *without respecting the time constraint the protocol implies*. Note that it will take at least several hours, or even days to complete.

Follow these steps:

- Go to the app directory containing the bootstrap script

  `# cd $SPLUNK_HOME/etc/apps/gsfb/bin/`

- Edit the script to add the `api_key` as well as others parameters under the `CONFIG` section of this script. Those parameters include for example the username and password to use to connect to Splunk KVStore to store the retrieved chunks. An example of configuration is given hereafter.

- Run the script in a terminal multiplexer like `screen`.

- Wait a few hours or days for the script to complete.

```
# screen
# python manual_bootstrap.py
(now, wait few hours or days for this script to complete)
```

bootstrap.py configuration example

```
##########
# CONFIG #
##########

# Lists to retrieve
lists = ['goog-malware-shavar', 'googpub-phish-shavar', 'goog-unwanted-shavar']

# Google API key
api_key   = "Bkae........mrsFG"

# KVStore parameters - usually only the username and the password have to be adjusted.
user     = 'admin'
password = 'mySecretAndSecurePassw0rd'
host = 'localhost'
port = 8089
```

Notes:

- This step is optional and can be skipped. However, doing so, the initial syncing of the lists could take multiple days or even weeks as Google's protocol indicates time delays before the next update can happen (usually around 30 minutes). This bootstrap script will not respect this delay and will ask for the next chunks as soon as the previous request is completed. This step will significantly reduce the time of the initial syncing but may still require a few days to complete (This is why its better to run it a screen-ed terminal).

- Because this script ignores the delay between two requests, Google could block the requests for a little while. In that case, only the `api_key` will be banned for a short period of time (usually few hours). During the development of this app - requiring multiple synchronizations - this particular situation was only faced once.
- In case of failure of this script (such as HTTP Error 400), simply ignore the error, wait few minutes and re-run the script (it will resume where it was before crashing).

4- Configure the Data Inputs.

At this point, if the bootstrap script was used, **it is mandatory to wait until it finishes before moving forward. In any case, there should be a Data Input *enabled* and the bootstrap script running**.

To configure the app, go under **Data Inputs > Google Safe Browsing** and adjust the parameters that fit your environment. The most important ones are your `api_ key` and the lists you want to retrieve.

Note:

- If the bootstrap step was skipped, the lists will update according to the protocol specifications and it could take weeks to fully synchronize the whole lists.

Using the Google Safe Browsing Splunk App

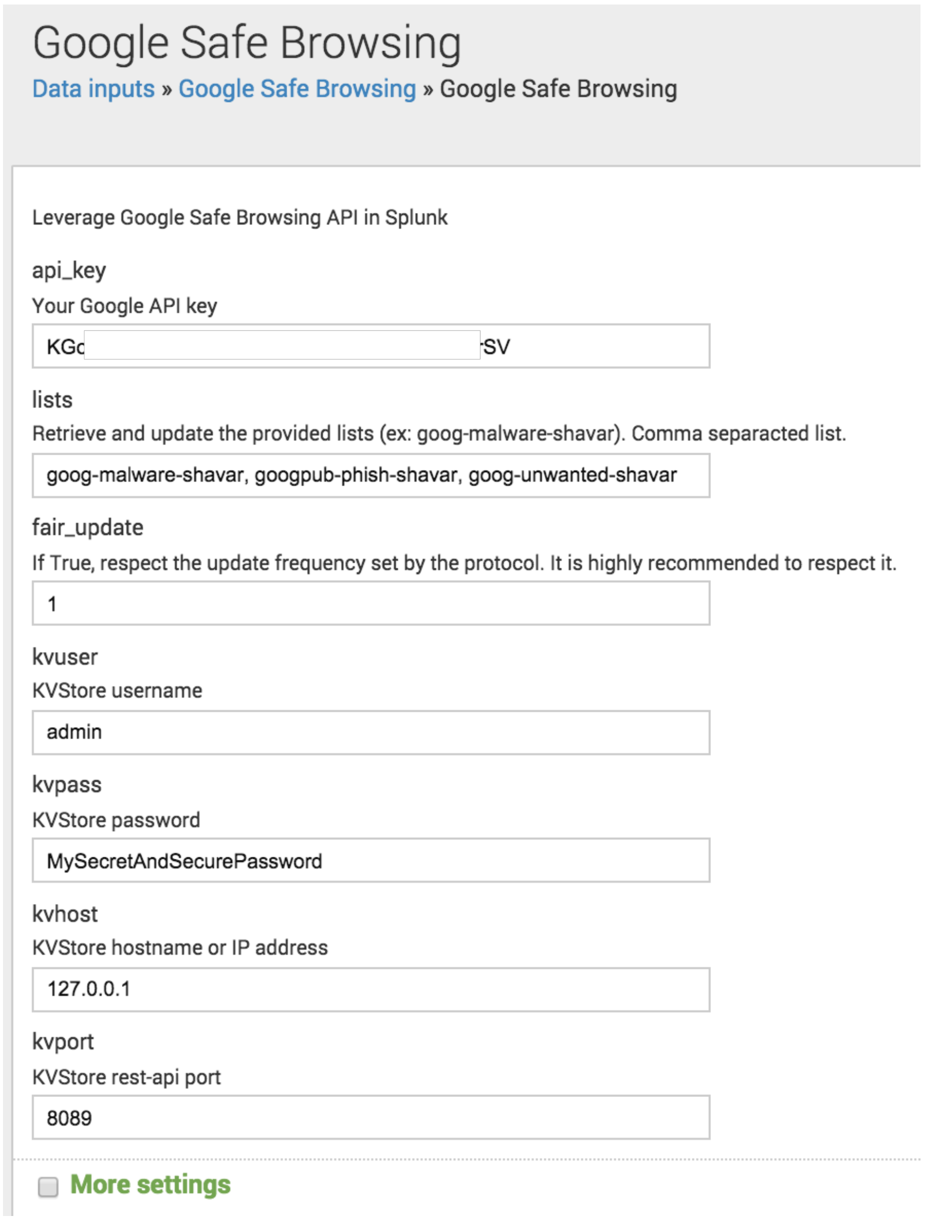
The Google Safe Browsing App for Splunk provides a set of atomic lookups that allow more flexibility. While those lookups have been designed to work together, one after the other, nothing prevents a search from using them for purposes other than in the context of the Google Safe Browsing app. For example, the canonicalize lookup (see below) could be used on all URLs to normalize them before searching for patterns like SQL Injection (this lookup takes care of the encoding, double-encoding, etc, that may appears in URLs).

1 - Canonicalized lookup

The purpose of this scripted lookup is to prepare any given URL according to Google's specifications. This preparation step is mandatory before computing the hashes. As output, the field `url_canonicalized` is created.

Examples:

Lookup call (using the field `cs_uri`):



Google Safe Browsing

Data inputs » Google Safe Browsing » Google Safe Browsing

Leverage Google Safe Browsing API in Splunk

api_key

Your Google API key

KGc[redacted]SV

lists

Retrieve and update the provided lists (ex: goog-malware-shavar). Comma separacted list.

goog-malware-shavar, googpub-phish-shavar, goog-unwanted-shavar

fair_update

If True, respect the update frequency set by the protocol. It is highly recommended to respect it.

1

kvuser

KVStore username

admin

kvpass

KVStore password

MySecretAndSecurePassword

kvhost

KVStore hostname or IP address

127.0.0.1

kvport

KVStore rest-api port

8089

Figure 2: Data Input Example

```
... | lookup canonicalize_lookup url as cs_uri
```

Or using the macro notation: `... | ‘canonicalize(cs_uri)‘`

An example of URL canonicalization:

- `Canonicalize("http://3279880203/blah#frag") = "http://195.127.0.11/blah"`

For more details, please refer to Google canonicalization specifications.

2 - Suffix Expression lookup

The purpose of this scripted lookup is to generate the possible permutations of the domain and the path of the input URL and to compute the SHA-256 hash of each of those permutations, as well as the hash prefixes (first 4-bytes of the hashes). The field `url_prefixes` is created as output and contains data represented as JSON which can be later parsed using the Splunk `spath` command.

Lookup call (using the field `url_canonicalized`):

```
... | lookup suffix_expression_lookup url as url_canonicalized
```

Or using the macro notation: `... | ‘suffix_expression(url_canonicalized)‘`

An example of the possible permutations for the input URL `http://a.b.c/1/2.html?param=1` :

```
a.b.c/1/2.html?param=1
a.b.c/1/2.html
a.b.c/
a.b.c/1/
b.c/1/2.html?param=1
b.c/1/2.html
b.c/
b.c/1/
```

For each of those permutations, the corresponding SHA-256 hash will be calculated.

For more details, please refer to Google Suffix Expression Lookup specifications.

3 - KV Store lookup for hash prefixes

The `prefixes` collection has the following fields:

- prefix (ex: ‘7ed8eda0’)
- prefix_length (ex: 4)
- chunk_key (ex: 557ec7f09fddc70c01696762)

To query this collection, one or more of these fields are required. A typical use case would be to provide as input the calculated prefixes by the Suffix Expression Lookup to retrieve the `chunk_key` which references a chunk (like a foreign key in RDBMS).

Lookup call (using the field `g_prefix`):

```
| lookup kv_prefixes_lookup prefix as g_prefix
```

Or using the macro notation: `... | ‘kv_prefixes(g_prefix)‘`

This lookup will basically tells you if the hash prefixes are known or not, but it will not tell you *how* they are known (add/sub chunks). Proceed with the KV Store lookup for chunks hereafter to know it.

4 - KV Store lookup for chunks

The `chunks` collection has the following fields:

- chunk_number (ex: 186791)
- chunk_type (‘add’ or ‘sub’)
- list_name (ex: ‘goopub-malware-shavar’)
- prefixes_count (ex: 35, the number of hash prefixes in that particular chunk).

To query this collection, provide one or more of those fields. A typical use case would be to provide as input the retrieved `chunk_key` by the `kv_prefixes_lookup` to retrieve the details (list_name, chunk_type, etc) of the corresponding chunks.

Lookup call (using the field `chunk_key`):

```
... | lookup kv_chunks_lookup _key as chunk_key
```

Or using the macro notation: `... | ‘kv_chunks(chunk_key)‘`

5 - When it all comes together

In the following example all logs are assumed Splunk's Common Information Model (CIM) compatible so will look for URLs in the field url:

To successfully leverage the previous lookups, a single macro has been created. In the following example all logs are assumed Splunk's Common Information Model (CIM) compatible so we will look for URLs in the field `url`.

```
url=* | fields url | ‘GoogleSafeBrowsing(url)‘
```

To have a more in-depth understanding of what the macro does, here is the detail of it:

```
(01) url=*
(02) | fields url
(03) | stats count by url
(04) | lookup canonicalize_lookup url as url
(05) | stats count by url_canonicalized
(06) | lookup suffix_expression_lookup url as url_canonicalized
(07) | spath input=url_prefixes
(08) | rename {}.gsfb_prefix AS g_prefix
(09) | rename {}.gsfb_hash AS g_hash
(10) | rename {}.gsfb_expression AS g_expr
(11) | lookup kv_prefixes_lookup prefix as g_prefix
(12) | search chunk_key=*
(13) | fields url_canonicalized chunk_key
(14) | lookup kv_chunks_lookup _key as chunk_key
(15) | eval tmp = mvfilter(match(chunk_type, "^add$"))
(16) | eval c = mvcount(tmp)
(17) | eval n_add = if(isnull(c), 0, c)
(18) | eval tmp = mvfilter(match(chunk_type, "^sub$"))
(19) | eval c = mvcount(tmp)
(20) | eval n_sub = if(isnull(c), 0, c)
(21) | eval chunk_type_sum = n_add - n_sub
(22) | search chunk_type_sum > 0
(23) | table url_canonicalized chunk_type list_name chunk_number chunk_key
```

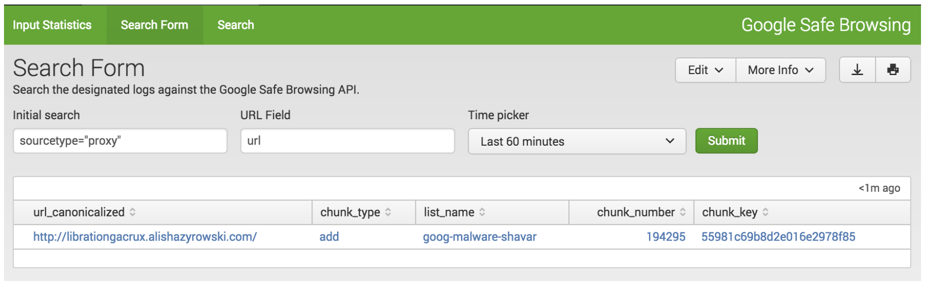
Query explanation:

- 01: Search for events having the field `url` set.
- 02: Filter out the fields and only keep the `url` field
- 03: Deduplicate the URL values

- 04: Canonicalize every URLs
- 05: Deduplicate the URLs now canonicalized
- 06: Compute the permutations and prefixes of each canonicalized URL.
- 07: Parse the results as JSON with `spath`.
- 8-10: Rename the fields parsed from previous point.
- 11: Look in the KVStore if the computed prefixes are known
- 12: Only keeps the results where the prefixes are known
- 13: Filter out the kept fields
- 14: Using the `chunk_key`, retrieve the chunks information (`list_name`, etc)
- 15-17: Count the number of 'add' chunks
- 18-20: Count the number of 'sub' chunks
- 21: Compute the difference between add and sub chunks in `chunk_type_sum`
- 22: Only keep results having more add chunks than sub chunks
- 23: Display the resulting fields in a table.

6 - Search form

The following search form is also available within the app.



Search Form

Search the designated logs against the Google Safe Browsing API.

Initial search: sourcetype="proxy"

URL Field: url

Time picker: Last 60 minutes

<1m ago

url_canonicalized	chunk_type	list_name	chunk_number	chunk_key
http://librationgacrux.alishazyrowski.com/	add	goog-malware-shavar	194295	55981c69b8d2e016e2978f85

Figure 3: Search Form

This form allows choosing the 'initial search' which, according to the previous example, would be here `url=* | fields url` and then specify the field to use (i.e.: the field containing URLs).

7 - Manually testing an URL

Here is a Splunk search that will check for a specific URL. Note that in June 2015 this URL was actually blacklisted by Google under the goopub-malware-shavar list, but it may not be the case at any other time.

```
| stats count
| eval u = "http://librationgacrux.alishazyrowski.com/"
| ‘GoogleSafeBrowsing(u)‘
```

Note that a workflow action has been created to run the above search directly from the raw events when the field `url` is present in your logs. This worflow action can be triggered either the event level or the field level.

8 - Full length hashes

As described in the protocol specifications, all the above steps are the result of comparing input URLs with a local database of 4-byte hash prefixes.

To guarantee the input URLs are actually listed by the Google Safe Browsing service, retrieve all the full length hashes corresponding to the hash prefix.

To do so, the following macro has been created:

```
... | lookup full_length_hashes_lookup hash as <hash_field> prefix_length as <prefix_length_
```

Or using the macro notation

```
... | ‘full_length_hashes(hash, prefix_length)‘
```

This lookup will create the field `confirmed_lists` which will contains all the lists where Google has listed this hash. In case there is no match, this field will be empty.

Moreover, a global macro has also been created to extend the previously presented `GoogleSafeBrowsing()` macro. The following macro will perform the same verifications (local checks of the prefixes) and will also retrieve the full length hashes.

```
... | GoogleSafeBrowsingFullHashCheck(u)
```

Notes:

- Because each hash as a limited lifetime (usually 10 minutes), the choice has been made to not store those full hashes in the KVStore. Each full length hash validation will require to query Google Safe Browsing API.
- An input stanza is required. This scripted lookup will automatically load the `local/inputs.conf` configuration file to retrieve your Google API Key.

9 - POST Lookup API

Google Safe Browsing also provides a 'Lookup API' dedicated to query the Google Safe Browsing API directly with the URLs without passing through the chunk's verification. This will send the URLs to Google service and it returns the classification.

More information about this specific API can be found here.

However, please note that a single API key can make requests for **up to 10,000 clients per 24-hour** period.

While its mostly outside of the perimeter of this app, the following scripted lookup has been created to perform some testing. This could be useful for example to cross validate matches with the local chunks.

```
... | lookup safe_browsing_lookup_api_lookup url as <url-field>
```

Or using the macro notation:

```
... | safe_browsing_lookup_api(url)
```

The created field `verdict` will contain the result like such as `malware` or `phishing` or `OK` if the URL has not been found in any of the Google's blacklists.