Well-Known URI spec per RFC5785

URI suffix: statements.json

Change controller: Google. Email: well-known-statements@google.com

Specification document(s): <link to the doc1 below>

Related information: <link to the doc2 below>

Doc1 (to be hosted publicly by Google): The “statements.json” Well-Known Resource Identifier

This document describes the use of URIs whose path component is “/.well-known/statements.json”.

Purpose

A URI with the path component “/.well-known/statements.json” is used by the AssetLinks protocol to identify one or more digital assets (such as web sites or mobile apps) that are related to the hosting web site in some fashion.

Details

Please refer to the Asset Links specification for details, specifically the section “Determining the Set of Reliable Statements”.

Example

```
[{ "namespace": "android_app",
   "package_name": "net.costingtons.mobileapp",
   "sha256_cert_fingerprints":
     ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:"
      "A0:83:42:E6:1D:BE:A8:8A:04:96:B2:3F:CF:44:E5"]}]
```

Doc2 (to be hosted publicly by Google): Asset Links Specification

Overview

Asset Links is a protocol to securely capture statements made by digital assets such as web sites or mobile apps about their relationship with other digital assets. Statements can reliably be attributed to the owners of the source assets.

Statement Model

A digital ***asset*** ***a*** is an identifiable and addressable online entity that typically provides some service or content.. Examples of assets are websites, mobile apps, and social media pages.

Ownership of an asset is defined by being able to control it and speak for it.

An asset owner may establish a ***relationship*** between the asset and another asset by making a statement about an intended relationship between the two. An example of a relationship is permission delegation. For example, the owner of a website (the webmaster) may delegate the ability the handle URLs to a particular mobile app. Relationships are considered public information.

A particular kind of relationship (like permission delegation) defines a binary relation on assets. The relation is not symmetric or transitive, nor is it antisymmetric or antitransitive.

A ***statement*** `S(r, a, b)` is an assertion that the relation r holds for the ordered pair of assets (a,b).
For example, taking r = “delegates permission to view user’s location”, a = New York Times mobile app, b = nytimes.com website, `S(r, a, b)` would be an assertion that “the New York Times mobile app delegates its ability to use the user’s location to the nytimes.com website”.

A statement `S(r, a, b)` is considered ***reliable*** if we have confidence that the statement is true; the exact criterion depends on the kind of statement, since some kinds of statements may be true on their face whereas others may require multiple parties to agree.

Data Types

AssetDescriptor

We name an asset using an **`AssetDescriptor`**, which is a JSON string. For the initial version of the API, we support only two types of assets, web sites and Android apps (in the future, others may be added but this does not affect the model).

Web site descriptors are as follows:

```
// if no port given, default port for scheme is assumed
{ "namespace": "web",
  "site": "http[s]://{fully-qualified domain}{:optional port}"
}
```

We will only support describing an entire domain at once. Note that only fully qualified domain names are permitted in the URL.

Android app descriptors are as follows:

```
{ "namespace": "android_app", "package_name": "[Java package name]",
  "sha256_cert_fingerprints": ["[SHA256 fingerprint of signing
cert]",
            "[additional allowed cert]", …]
}
```

The sha256_cert_fingerprints is a list of colon-separated hex strings. Note that the list is syntactic sugar: since only one certificate is required to identify an android app, an asset descriptor with two fingerprints is equivalent to two asset descriptors with one fingerprint each.
Full example of an Android app descriptor:

```
{ "namespace": "android_app", "package_name": "com.costingtons.app",
  "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:"
     "A0:83:42:E6:1D:BE:A8:8A:04:96:B2:3F:CF:44:E5"]}
```

Notably, we do not refer to any particular version of the app: a phone-based implementation may match the installed version, while a central service may use the latest version in the Play store.
iOS App asset descriptors are as follows:

```
{ "namespace": "ios_app", "appid": "585027354" }
```

Relation

A **Relation** describes the nature of a statement, and consists of a *kind* and a *detail*.
The set of kinds is enumerated by the API:

Kind	Description / Reliability determination
delegate_permission	The detail field specifies which permission to delegate. A statement involving this relation does not constitute a *requirement* to do the delegation, just a *permission* to do so. How to determine reliability: A statement of this kind is reliable if it's made by the owner of the asset doing the delegating.

We anticipate adding other kinds in the future, but are being deliberately cautious in doing so.

The detail field is a lowercase alphanumeric string with underscores and periods allowed (matching the regex [a-z0-9_.]+), but otherwise unstructured. We define some common values to avoid unnecessary fragmentation of the API:

Pre-defined kind / detail	Description
delegate_permission / "common.handle_all_urls"	Delegates the ability to handle all URLs that the source asset can handle

If defining custom detail strings, we encourage the use of Java-style scoping, e.g, "com.google.app.detailstring". A special detail string of "*" may be used to identify all relations within the specified kind.

Statement

A **Statement** asserts a relation between an ordered pair of assets. The asset whose owner is making the statement is implicitly the first one (the "source") in the relation; we may in the future allow the source asset to be specified explicitly. The statement is a JSON string with the following structure:

```
{
  "relation": ["{kind}/[{detail} | *]", ...],
  "target" : { AssetDescriptor }
}
```

As described above, a "*" value for the detail field implies a statement made about all possible detail strings for the specified kind.

Example statement being made by a website that it delegates the ability to handle URLs to a particular app:

```
{
  "relation": ["delegate_permission/common.handle_all_urls"],
```

```
  "target" : { "namespace": "android_app",
               "package_name": "com.mydomain.app",
               "sha256_cert_fingerprints": ["[cert hash]"] }
}
```

To allow for easier scaling and central management, we also allow a `Statement` which points to another file containing a list of Statements that should be logically included:

```
{
  "include": "https://[url of file to include, with FQDN]"
}
```

If the source asset's delivery mechanism is secure (e.g., HTTPS or signed Android APKs), the include file must be served over SSL. Include files are only processed if the HTTP status code is 200; otherwise the content of that include file is discarded. Specifically, HTTP 30x redirects will not be followed.

In practice, we expect that direct statements will suffice for small numbers of assets, but that include-files will be more common when dealing with hundreds of assets whose relationships are better managed centrally.

Statement List

A Statement List is a JSON array with any number of statements.

Determining the Set of Reliable Statements

As defined above, *reliable* statements are those intentionally made by asset owners. For each asset type, we identify a location which we believe can only be controlled by the asset's owner. We treat all statements in that location, as well as all statements pulled in via the inclusion mechanism, as being made by the asset owner. The validity period of each statement is generally determined by the TTL of the underlying data, though a particular implementation may also impose a lower and upper bound.

Website:

Granularity: A website asset is characterized by (scheme, domain, and port). All paths and query strings are assumed to be a part of the same website. Both http and https schemes are supported.
Statement location: For website `<scheme>://<domain>:<port>/`, we consider the file at `<scheme>://<domain>:<port>/.well-known/statements.json` to contain a Statement list in JSON format as defined above. The file should be served with the media type "application/json". Any response besides HTTP 200 is treated as an error, and will result in an empty statements list.

Android app:

Granularity: An Android app is characterized by (package name, signing cert).

Statement location: Android app manifests are protected by the APK signature whose signing key is by definition controlled by the owner of the app.

In AndroidManifest.xml:

```
<manifest>
  <application>
    ...
    <meta-data android:name="asset_statements"
android:resource="@string/asset_statements" />
  </application>
</manifest>
```

In res/values/strings.xml:

```
<resources>
    ...
    <string name="asset_statements">
        {JSON array of statements}
    </string>
</resources>
```

iOS app:

Granularity: An iOS app is identified by its App ID. For example, the Google Maps app has the App ID 585027354.

Statement location: As part of their signed bundle, iOS apps contain an Info.plist file. Asset Link statements are placed inside this file:

```
<key>AssetLinkManifest</key>
<string>
  {JSON array of statements}
</string>
```