Network Working Group
Internet Draft
Intended status: Standards Track
Expires: November 26, 2013

Paul E. Jones
Gonzalo Salgueiro
Cisco Systems
Joseph Smarr
Google

May 26, 2013

WebFinger
draft-ietf-appsawg-webfinger-14.txt

Abstract

This specification defines the WebFinger protocol, which can be used to discover information about people or other entities on the Internet using standard HTTP methods. WebFinger discovers information for a URI that might not be usable as a locator otherwise, such as account or email URIs.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

WebFinger is used to discover information about people or other entities on the Internet that are identified by a URI [6] or IRI [7] using standard Hypertext Transfer Protocol (HTTP) [2] methods over a secure transport [14]. A WebFinger resource returns a JavaScript Object Notation (JSON) [5] object describing the entity that is queried. The JSON object is referred to as the JSON Resource Descriptor (JRD).

For a person, the kinds of information that might be discoverable via WebFinger include a personal profile address, identity service,

telephone number, or preferred avatar. For other entities on the Internet, a WebFinger resource might return JRDs containing link relations [10] that enable a client to discover, for example, the

that a printer can print in color on A4 paper, the physical location of a server, or other static information.

Information returned via WebFinger might be for direct human consumption (e.g., looking up someone's phone number), or it might be used by systems to help carry out some operation (e.g., facilitate logging into a web site by determining a user's identity service). The information is intended to be static in nature and, as such, WebFinger is not intended to be used to return dynamic information like the temperature of a CPU or the current toner level in a laser printer.

The WebFinger protocol is designed to be used across many applications. Applications that wish to utilize WebFinger will need to specify properties, titles, and link relation types that are appropriate for the application. Further, applications will need to define the appropriate URI scheme to utilize for the query target.

Use of WebFinger is illustrated in the examples in Section 3 and described more formally in Section 4.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

WebFinger makes heavy use of "Link Relations". A Link Relation is an attribute-and-value pair in which the attribute identifies the type of relationship between the linked entity or resource and the information specified in the value. In Web Linking [4], the link relation is represented using an HTTP entity-header of "Link", where the "rel" attribute specifies the type of relationship and the "href" attribute specifies the information that is linked to the entity or resource. In WebFinger, the same concept is represented using a JSON array of "links" objects, where each member named "rel" specifies the type of relationship and each member named "href" specifies the information that is linked to the entity or resource. Note that WebFinger narrows the scope of a link relation beyond what is defined for Web Linking by stipulating that the value of the "rel" member needs to be either a single IANA-registered link relation type [10] or a URI [6].

3. Example Uses of WebFinger

This non-normative section shows a few sample uses of WebFinger.

3.1. Locating a User's Blog

Assume you receive an email from Bob and he refers to something he posted on his blog, but you do not know where Bob's blog is located.

It would be simple to discover the address of Bob's blog if he made that information available via WebFinger.

Assume your email client can discover the blog for you. After receiving the message from Bob (bob@example.com), your email client performs a WebFinger query either automatically or at your command. (Please refer to Section 8.2 for user privacy considerations and Section 8.3 for abuse considerations, particularly when considering any kind of automated query feature.) It does so by issuing the following HTTPS [14] query to example.com:

```
GET /.well-known/webfinger?
                        resource=acct%3Abob%40example.com HTTP/1.1
Host: example.com
```

The server might then respond with a message like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "acct:bob@example.com",
  "aliases" :
  [
    "http://www.example.com/~bob/"
  ],
  "properties" :
  {
      "http://example.com/ns/role/" : "employee"
  },
  "links" :
  [
    {
      "rel" : "http://webfinger.example/rel/avatar",
      "type" : "image/jpeg",
      "href" : "http://www.example.com/~bob/bob.jpg"
    },
    {
      "rel" : "http://webfinger.example/rel/profile-page",
      "href" : "http://www.example.com/~bob/"
    },
    {
      "rel" : "http://webfinger.example/rel/blog",
      "type" : "text/html",
      "href" : "http://blogs.example.com/bob/",
      "titles" :
      {
```

```
              "en-us" : "The Magical World of Bob",
              "fr" : "Le Monde Magique de Bob"
            }
          },
```

```
        {
          "rel" : "http://webfinger.example/rel/businesscard",
          "href" : "https://www.example.com/~bob/bob.vcf"
        }
      ]
    }
```

Note the assumption made in the above example is that there is an "acct" URI for the given "mailto" URI. This may not always be the case.

The email client would take note of the link relation in the above JRD that refers to Bob's blog. The blog's URI would then be presented to you so that you could then visit his blog. The email client might also note that Bob has published an avatar link relation and use that picture to represent Bob inside the email client. Lastly, the client might automatically retrieve the data located at the URI specified by the "businesscard" link relation (which might be a vcard [16]) to update the information about Bob in its internal address book.

In the above example, an "acct" URI [8] is used in the query, though any valid alias for the user might also be used. See Section 4.5 for more information on WebFinger and URIs.

An alias is a URI that is different from the "subject" URI, yet identifies the same entity. In the above example, there is one "http" alias returned, though there might have been more than one. Had the "http:" URI shown as an alias been used to query for information about Bob, the query would have appeared as:

```
  GET /.well-known/webfinger?
           resource=http%3A%2F%2Fwww.example.com%2F~bob%2F HTTP/1.1
  Host: www.example.com
```

Note that the host queried in this example is different than for the acct URI example, since the URI refers to a different host. Either this host would provide a response, or it would redirect the client to another host (e.g., redirect back to example.com). Either way, the response would have been substantially the same, with the subject and alias information changed as necessary.

3.2. Identity Provider Discovery for OpenID Connect

Suppose Carol wishes to authenticate with a web site she visits using OpenID Connect [18]. She would provide the web site with her OpenID Connect identifier, say carol@example.com. The visited web site would perform a WebFinger query looking for the OpenID Connect Provider. Since the site is interested in only one particular link

relation, the WebFinger resource might utilize the "rel" parameter as described in Section 4.3:

```
GET /.well-known/webfinger?
       resource=acct%3Acarol%40example.com&
       rel=http%3A%2F%2Fopenid.net%2Fspecs%2Fconnect%2F1.0%2Fissuer
       HTTP/1.1
Host: example.com
```

The server might respond like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "acct:carol@example.com",
  "links" :
  [
    {
      "rel" : "http://openid.net/specs/connect/1.0/issuer",
      "href" : "https://openid.example.com"
    }
  ]
}
```

Since the "rel" parameter only serves to filter the link relations returned by the resource, other name/value pairs in the response, including any aliases or properties, would be returned. Also, since support for the "rel" parameter is not guaranteed, the client must not assume the "links" array will contain only the requested link relation.

3.3. Auto-Configuration of Email Clients

WebFinger could be used to auto-provision an email client with basic configuration data. Suppose that sue@example.com wants to configure her email client. Her email client might issue the following query:

```
GET /.well-known/webfinger?
         resource=mailto%3Asue%40example.com HTTP/1.1
Host: example.com
```

The returned resource representation would contain entries for the various protocols, transport options, and security options. If there are multiple options, the resource representation might include a link relation for each of the valid options, and the client or Sue might select which option to choose. Since JRDs list link relations in a specific order, then the most-preferred choices could be presented first. Consider this response:

```
HTTP/1.1 200 OK
```

```
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json
```

```
{
  "subject" : "mailto:sue@example.com",
  "links" :
  [
    {
      "rel" : "http://webfinger.example/rel/smtp-server",
      "properties" :
      {
        "http://webfinger.example/email/host" : "smtp.example.com",
        "http://webfinger.example/email/port" : "587",
        "http://webfinger.example/email/login-required" : "yes",
        "http://webfinger.example/email/transport" : "starttls"
      }
    },
    {
      "rel" : "http://webfinger.example/rel/imap-server",
      "properties" :
      {
        "http://webfinger.example/email/host" : "imap.example.com",
        "http://webfinger.example/email/port" : "993",
        "http://webfinger.example/email/transport" : "ssl"
      }
    }
  ]
}
```

In this example, you can see that the WebFinger resource representation advertises an SMTP service and an IMAP service. In this example, the "href" entries associated with the link relation are absent. This is valid when there is no additional reference that needs to be made.

3.4. Retrieving Device Information

As another example, suppose there are printers on the network and you would like to check a particular printer identified by the URI device:p1.example.com to see if it can print in color on A4 paper. While the "device" URI scheme is not presently specified, we use it here for illustrative purposes.

Following the procedures similar to those above, a query may be issued to get link relations specific to this URI like this:

```
GET /.well-known/webfinger?
                   resource=device%3Ap1.example.com HTTP/1.1
Host: p1.example.com
```

The link relations that are returned for a device may be quite

different than those for user accounts. Perhaps we may see a response like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "device:p1.example.com",
  "links" :
  [
    {
      "rel" : "http://webfinger.example/rel/tipsi",
      "href" : "http://192.168.1.5/npap/"
    }
  ]
}
```

While this example is fictitious, you can imagine that perhaps the Transport Independent, Printer/System Interface [17] may be enhanced with a web interface enabling a device that understands the TIP/SI web interface specification to query for printer capabilities.

4. WebFinger Protocol

The WebFinger protocol is used to request information about an entity identified by a query target (a URI). The client can optionally specify one or more link relation types for which it would like to receive information.

A WebFinger request is an HTTPS request to a WebFinger resource. A WebFinger resource is a well-known URI [3] using the HTTPS scheme, constructed along with the required query target and optional link relation types. WebFinger resources MUST NOT be served with any other URI scheme (such as HTTP).

A WebFinger resource is always given a query target, which is another URI that identifies the entity whose information is sought. GET requests to a WebFinger resource convey the query target in the "resource" parameter in the WebFinger URI's query string; see Section 4.1 for details.

The host to which a WebFinger query is issued is significant. If the query target contains a "host" portion (Section 3.2.2 of RFC 3986), then the host to which the WebFinger query is issued MUST be the same as the "host" portion of the query target, unless the client receives instructions through some out-of-band mechanism to send the query to another host. If the query target does not contain a "host" portion, then the client MAY choose a host to which it directs the query using additional information it has.

The path component of a WebFinger URI MUST be the well-known path

"/.well-known/webfinger". A WebFinger URI MUST contain a query component that encodes the query target and optional link relation types as specified in Section 4.1.

The WebFinger resource returns a JSON Resource Descriptor (JRD) as the resource representation to convey information about an entity on the Internet. Also, the Cross-Origin Resource Sharing (CORS) [9] specification is utilized to facilitate queries made via a web browser.

4.1. Constructing the Query Component of the Request URI

A WebFinger URI MUST contain a query component (see Section 3.4 of RFC 3986). The query component MUST contain a "resource" parameter and MAY contain one or more "rel" parameters. The "resource" parameter MUST contain the query target (URI) and the "rel" parameters MUST contain encoded link relation types according to the encoding described in this section.

To construct the query component, the client performs the following steps. First, each parameter value is percent-encoded, as per Section 2.1 of RFC 3986. The encoding is done to conform to the query production in Section 3.4 of that specification, with the addition that any instances of the "=" and "&" characters within the parameter values are also percent-encoded. Next, the client constructs a string to be placed in the query component by concatenating the name of the first parameter together with an equal sign ("=") and the percent-encoded parameter value. For any subsequent parameters, the client appends an ampersand ("&") to the string, the name of the next parameter, an equal sign, and the parameter value. The client MUST NOT insert any spaces while constructing the string. The order in which the client places each attribute-and-value pair within the query component does not matter in the interpretation of the query component.

4.2. Performing a WebFinger Query

A WebFinger client issues a query using the GET method to the well-known [3] resource identified by the URI whose path component is "/.well-known/webfinger" and whose query component MUST include the "resource" parameter exactly once and set to the value of the URI for which information is being sought. If the "resource" parameter is absent or malformed, the WebFinger resource MUST indicate that the request is bad as per Section 10.4.1 of RFC 2616 [2].

A client MUST query the WebFinger resource using HTTPS only. If the client determines that the resource has an invalid certificate, the resource returns a 4xx or 5xx status code, or the HTTPS connection cannot be established for any reason, then the client MUST accept that the WebFinger query has failed and MUST NOT attempt to reissue the WebFinger request using HTTP over a non-secure connection.

A WebFinger resource MUST return a JRD as the representation for the resource if the client requests no other supported format explicitly via the HTTP "Accept" header. The client MAY include the "Accept" header to indicate a desired representation; representations other

than JRD might be defined in future specifications. The WebFinger resource MUST silently ignore any requested representations that it does not understand and support. The media type used for the JSON Resource Descriptor (JRD) is "application/jrd+json" (see Section 9.2).

A WebFinger resource MAY redirect the client; if it does, the redirection MUST only be to an "https" URI and the client MUST perform certificate validation again when redirected.

A WebFinger resource can include cache validators in a response to enable conditional requests by the client and/or expiration times as per Section 13 of RFC 2616.

4.3. The "rel" Parameter

When issuing a request to a WebFinger resource, the client MAY utilize the "rel" parameter to request only a subset of the information that would otherwise be returned without the "rel" parameter. When the "rel" parameter is used and accepted, only the link relation types that match the link relation types provided via the "rel" parameter are included in the array of links returned in the JRD. If there are no matching link relation types defined for the resource, the "links" array in the JRD will either be absent or empty. All other information present in a resource descriptor remains present, even when "rel" is employed.

The "rel" parameter MAY be included multiple times in order to request multiple link relation types.

The purpose of the "rel" parameter is to return a subset of "link relation objects" (see Section 4.4.4) that would otherwise be returned in the resource descriptor. Use of the parameter might reduce processing requirements on either the client or server, and it might also reduce the bandwidth required to convey the partial resource descriptor, especially if there are numerous link relation values to convey for a given "resource" value.

WebFinger resources SHOULD support the "rel" parameter. If the resource does not support the "rel" parameter, it MUST ignore the parameter and process the request as if no "rel" parameter values were present.

The following example presents the same example as found in Section 3.1, but uses the "rel" parameter to select two link relations:

```
GET /.well-known/webfinger?
       resource=acct%3Abob%40example.com&
       rel=http%3A%2F%2Fwebfinger.example%2Frel%2Fprofile-page&
```

```
        rel=http://webfinger.example/rel/businesscard HTTP/1.1
Host: example.com
```

In this example, the client requests the link relations of type "http://webfinger.example/rel/profile-page" and "http://webfinger.example/rel/businesscard". The server then responds with a message like this:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/jrd+json

{
  "subject" : "acct:bob@example.com",
  "aliases" :
  [
    "http://www.example.com/~bob/"
  ],
  "properties" :
  {
      "http://example.com/ns/role/" : "employee"
  },
  "links" :
  [
    {
      "rel" : "http://webfinger.example/rel/profile-page",
      "href" : "http://www.example.com/~bob/"
    },
    {
      "rel" : "http://webfinger.example/rel/businesscard",
      "href" : "http://www.example.com/~bob/bob.vcf"
    }
  ]
}
```

As you can see in the response, the resource representation contains only the link relations requested by the client, but the other parts of the JRD are still present.

4.4. The JSON Resource Descriptor (JRD)

The JSON Resource Descriptor (JRD), originally introduced in RFC 6415 [19] and based on the Extensible Resource Descriptor (XRD) format [20], is a JSON object that comprises the following name/value pairs:

- o subject
- o aliases
- o properties
- o links

The member "subject" is a name/value pair whose value is a string,

"aliases" is an array of strings, "properties" is an object comprising name/value pairs whose values are strings, and "links" is an array of objects that contain link relation information.

When processing a JRD, the client MUST ignore any unknown member and not treat the presence of an unknown member as an error.

Below, each of these members of the JRD is described in more detail.

4.4.1. subject

The value of the "subject" member is a URI that identifies the entity that the JRD describes.

The "subject" value returned by a WebFinger resource MAY differ from the value of the "resource" parameter used in the client's request. This might happen, for example, when the subject's identity changes (e.g., a user moves his or her account to another service) or when the resource prefers to express URIs in canonical form.

The "subject" member SHOULD be present in the JRD.

4.4.2. aliases

The "aliases" array is an array of zero or more URI strings that identify the same entity as the "subject" URI. Each URI must be an absolute URI.

The "aliases" array is OPTIONAL in the JRD.

4.4.3. properties

The "properties" object comprises zero or more name/value pairs whose names are absolute URIs and whose values are strings or null. Properties are used to convey additional information about the subject of the JRD. As an example, consider this use of "properties":

```
"properties" : { "http://webfinger.example/ns/name" : "Bob Smith" }
```

The "properties" member is OPTIONAL in the JRD.

4.4.4. links

The "links" array has any number of member objects, each of which represents a link [4]. Each of these link objects can have the following members:

- o rel
- o type
- o href
- o titles
- o properties

The "rel" and "href" members are strings representing the link's relation type and the target IRI, respectively. The context of the link is the "subject" (see Section 4.4.1).

The "type" member is a string indicating what the media type of the result of dereferencing the link ought to be.

The order of elements in the "links" array indicates an order of preference. Thus, if there are two or more link relations having the same "rel" value, the first link relation would indicate the user's preferred link.

The "links" array is OPTIONAL in the JRD.

Below, each of the members of the objects found in the "links" array is described in more detail. Each object in the "links" array, referred to as a "link relation object", is completely independent from any other object in the array; any requirement to include a given member in the link relation object refers only to that particular object.

4.4.4.1. rel

The value of the "rel" member is a string that is either an absolute URI or a registered relation type [10] (see RFC 5988 [4]). The value of the "rel" member MUST contain exactly one URI or registered relation type. The URI or registered relation type identifies the type of the link relation.

The other members of the object have meaning only once the type of link relation is understood. In some instances, the link relation will have associated semantics enabling the client to query for other resources on the Internet. In other instances, the link relation will have associated semantics enabling the client to utilize the other members of the link relation object without fetching additional external resources.

URI link relation type values are compared using the "Simple String Comparison" algorithm of section 6.2.1 of RFC 3986 [6].

The "rel" member MUST be present in the link relation object.

4.4.4.2. type

The value of the "type" member is a string that indicates the media type [11] of the target resource (see RFC 6838 [12]).

The "type" member is OPTIONAL in the link relation object.

4.4.4.3. href

The value of the "href" member is a string that contains a URI pointing to the target resource.

The "href" member is OPTIONAL in the link relation object.

4.4.4.4. titles

The "titles" object comprises zero or more name/value pairs whose name is a language tag [13] or the string "und". The string is human-readable and describes the link relation. More than one title for the link relation MAY be provided for the benefit of users who utilize the link relation and, if used, a language identifier SHOULD be duly used as the name. If the language is unknown or unspecified, then the name is "und".

A JRD SHOULD NOT include more than one title identified with the same language tag (or "und") within the link relation object. Meaning is undefined if a link relation object includes more than one title named with the same language tag (or "und"), though this MUST NOT be treated as an error. A client MAY select whichever title or titles it wishes to utilize.

Here is an example of the titles object:

```
"titles" :
{
  "en-us" : "The Magical World of Bob",
  "fr" : "Le Monde Magique de Bob"
}
```

The "titles" member is OPTIONAL in the link relation object.

4.4.4.5. properties

The "properties" object within the link relation object comprises zero or more name/value pairs whose names are absolute URIs and whose values are strings or null. Properties are used to convey additional information about the link relation. As an example, consider this use of "properties":

```
"properties" : { "http://webfinger.example/mail/port" : "993" }
```

The "properties" member is OPTIONAL in the link relation object.

4.5. WebFinger and URIs

WebFinger requests include a "resource" parameter (see Section 4.1)

specifying the URI of an account, device, or other entity. WebFinger is neutral regarding the scheme of such a URI: it could be an "acct"

URI [7], an "http" or "https" URI, a "mailto" URI [21], or some other scheme.

To perform a WebFinger lookup on an account specific to the host being queried, use of the "acct" URI scheme is recommended, since it explicitly identifies a generic user account that is not necessarily bound to a specific protocol. Further, the "acct" URI scheme is not associated with other protocols as, by way of example, the "mailto" URI scheme is associated with email. Since not every host offers email service, using the "mailto" URI scheme is not ideal for identifying user accounts on all hosts. That said, use of the "mailto" URI scheme would be ideal for use with WebFinger to discover mail server configuration information for a user.

5. Cross-Origin Resource Sharing (CORS)

WebFinger resources might not be accessible from a web browser due to "Same-Origin" policies. The current best practice is to make resources available to browsers through Cross-Origin Resource Sharing (CORS) [9], and servers MUST include the Access-Control-Allow-Origin HTTP header in responses. Servers SHOULD support the least restrictive setting by allowing any domain access to the WebFinger resource:

```
Access-Control-Allow-Origin: *
```

There are cases where defaulting to the least restrictive setting is not appropriate, for example a server on an intranet that provides sensitive company information SHOULD NOT allow CORS requests from any domain, as that could allow leaking of that sensitive information. A server that wishes to restrict access to information from external entities SHOULD use a more restrictive Access-Control-Allow-Origin header.

6. Access Control

As with all web resources, access to the WebFinger resource could require authentication. Further, failure to provide required credentials might result in the server forbidding access or providing a different response than had the client authenticated with the server.

Likewise, a WebFinger resource MAY provide different responses to different clients based on other factors, such as whether the client is inside or outside a corporate network. As a concrete example, a query performed on the internal corporate network might return link relations to employee pictures, whereas link relations for employee pictures might not be provided to external entities.

Further, link relations provided in a WebFinger resource representation might point to web resources that impose access restrictions. For example, the aforementioned corporate server may

provide both internal and external entities with URIs to employee pictures, but further authentication might be required in order for the client to access the picture resources if the request comes from outside the corporate network.

The decisions made with respect to what set of link relations a WebFinger resource provides to one client versus another and what resources require further authentication, as well as the specific authentication mechanisms employed, are outside the scope of this document.

7. Hosted WebFinger Services

As with most services provided on the Internet, it is possible for a domain owner to utilize "hosted" WebFinger services. By way of example, a domain owner might control most aspects of their domain, but use a third-party hosting service for email. In the case of email, MX records identify mail servers for a domain. An MX record points to the mail server to which mail for the domain should be delivered. It does not matter to the sending mail server whether those MX records point to a server in the destination domain or a different domain.

Likewise, a domain owner might utilize the services of a third party to provide WebFinger services on behalf of its users. Just as a domain owner was required to insert MX records into DNS to allow for hosted email serves, the domain owner is required to redirect HTTP queries to its domain to allow for hosted WebFinger services.

When a query is issued to the WebFinger resource, the web server MUST return a response with a redirection status code that includes a Location header pointing to the location of the hosted WebFinger service URI. This WebFinger service URI does not need to point to the well-known WebFinger location on the hosting service provider server.

As an example, assume that example.com's WebFinger services are hosted by wf.example.net. Suppose a client issues a query for acct:alice@example.com like this:

```
GET /.well-known/webfinger?
              resource=acct%3Aalice%40example.com HTTP/1.1
Host: example.com
```

The server might respond with this:

```
HTTP/1.1 307 Temporary Redirect
Access-Control-Allow-Origin: *
Location: https://wf.example.net/example.com/webfinger?
```

```
                    resource=acct%3Aalice%40example.com
```

The client can then follow the redirection, re-issuing the request to the URI provided in the Location header. Note that the server will include any required URI parameters in the Location header value, which could be different than the URI parameters the client originally used.

8. Security Considerations

8.1. Transport-Related Issues

Since this specification utilizes Cross-Origin Resource Sharing (CORS) [9], all of the security considerations applicable to CORS are also applicable to this specification.

The use of HTTPS is REQUIRED to ensure that information is not modified during transit. It should be appreciated that in environments where a web server is normally available, there exists the possibility that a compromised network might have its WebFinger resource operating on HTTPS replaced with one operating only over HTTP. As such, clients MUST NOT issue queries over a non-secure connection.

Clients MUST verify that the certificate used on an HTTPS connection is valid (as defined in [14]) and accept a response only if the certificate is valid.

8.2. User Privacy Considerations

Service providers and users should be aware that placing information on the Internet means that any user can access that information and WebFinger can be used to make it even easier to discover that information. While WebFinger can be an extremely useful tool for discovering one's avatar, blog, or other personal data, users should understand the risks, too.

Systems or services that expose personal data via WebFinger MUST provide an interface by which users can select which data elements are exposed through the WebFinger interface. For example, social networking sites might allow users to mark certain data as "public" and then utilize that marking as a means of determining what information to expose via WebFinger. The information published via WebFinger would thus comprise only the information marked as public by the user. Further, the user has the ability to remove information from publication via WebFinger by removing this marking.

WebFinger MUST NOT be used to provide any personal data unless publishing that data via WebFinger by the relevant service was explicitly authorized by the person whose information is being shared. Publishing one's personal data within an access-controlled

or otherwise limited environment on the Internet does not equate to providing implicit authorization of further publication of that data via WebFinger.

The privacy and security concerns with publishing personal data via WebFinger are worth emphasizing again with respect to personal data that might reveal a user's current context (e.g., the user's location). The power of WebFinger comes from providing a single place where others can find pointers to information about a person, but service providers and users should be mindful of the nature of that information shared and the fact that it might be available for the entire world to see. Sharing location information, for example, would potentially put a person in danger from any individual who might seek to inflict harm on that person.

Users should be aware of how easily personal data one might publish can be used in unintended ways. In one study relevant to WebFinger-like services, Balduzzi et al. [22] took a large set of leaked email addresses and demonstrated a number of potential privacy concerns, including the ability to cross-correlate the same user's accounts over multiple social networks. The authors also describe potential mitigation strategies.

The easy access to user information via WebFinger was a design goal of the protocol, not a limitation. If one wishes to limit access to information available via WebFinger, such as WebFinger resources for use inside a corporate network, the network administrator needs to take necessary measures to limit access from outside the network. Using standard methods for securing web resources, network administrators do have the ability to control access to resources that might return sensitive information. Further, a server can be employed in such a way as to require authentication and prevent disclosure of information to unauthorized entities.

8.3. Abuse Potential

Service providers should be mindful of the potential for abuse using WebFinger.

As one example, one might query a WebFinger server only to discover whether a given URI is valid or not. With such a query, the person may deduce that an email identifier is valid, for example. Such an approach could help spammers maintain a current list of known email addresses and to discover new ones.

WebFinger could be used to associate a name or other personal data with an email address, allowing spammers to craft more convincing email messages. This might be of particular value in phishing attempts.

It is RECOMMENDED that implementers of WebFinger server software take steps to mitigate abuse, including malicious over-use of the server

and harvesting of user information. Although there is no mechanism that can guarantee that publicly-accessible WebFinger databases won't be harvested, rate-limiting by IP address will prevent or at least dramatically slow harvest by private individuals without access to

botnets or other distributed systems. The reason these mitigation strategies are not mandatory is that the correct choice of mitigation strategy (if any) depends greatly on the context. Implementers should not construe this as meaning that they do not need to consider whether to use a mitigation strategy, and, if so, what strategy to use.

WebFinger client developers should also be aware of potential abuse by spammers or those phishing for information about users. As an example, suppose a mail client was configured to automatically perform a WebFinger query as discussed in the example in Section 3.1. If a spammer sent an email using a unique identifier in the 'From' header, then when the WF query was performed the spammer would be able to associate the request with a particular user's email address. This would provide information to the spammer, including the user's IP address, the fact the user just checked email, what kind of WebFinger client the user utilized, and so on. For this reason, it is strongly advised that clients not perform WebFinger queries unless authorized by the user to do so.

8.4. Information Reliability

A WebFinger resource has no means of ensuring that information provided by a user is accurate. Likewise, neither the resource nor the client can be absolutely guaranteed that information has not been manipulated either at the server or along the communication path between the client and server. Use of HTTPS helps to address some concerns with manipulation of information along the communication path, but it clearly cannot address issues where the resource provided incorrect information, either due to being provided false information or due to malicious behavior on the part of the server administrator. As with any information service available on the Internet, users should be wary of information received from untrusted sources.

9. IANA Considerations

9.1. Well-Known URI

This specification registers the "webfinger" well-known URI in the Well-Known URI Registry as defined by [3].

URI suffix: webfinger

Change controller: IETF

Specification document(s): RFC XXXX

Related information: The query to the WebFinger resource will

include one or more parameters in the query string; see Section 4.1 of RFCXXXX. Resources at this location are able to return a JSON Resource Descriptor (JRD) as described in Section 4.4 of RFCXXXX.

[RFC EDITOR: Please replace "XXXX" references in this section and the following section with the number for this RFC.]

9.2. JSON Resource Descriptor (JRD) Media Type

This specification registers the media type application/jrd+json for use with WebFinger in accordance with media type registration procedures defined in [12].

Type name: application

Subtype name: jrd+json

Required parameters: N/A

Optional parameters: N/A

In particular, because RFC 4627 already defines the character encoding for JSON, no "charset" parameter is used.

Encoding considerations: See RFC 6839, section 3.1.

Security considerations:

The JSON Resource Descriptor (JRD) is a JavaScript Object Notation (JSON) object. It is a text format that must be parsed by entities that wish to utilize the format. Depending on the language and mechanism used to parse a JSON object, it is possible for an attacker to inject behavior into a running program. Therefore, care must be taken to properly parse a received JRD to ensure that only a valid JSON object is present and that no JavaScript or other code is injected or executed unexpectedly.

Interoperability considerations:

This media type is a JavaScript Object Notation (JSON) object and can be consumed by any software application that can consume JSON objects.

Published specification: RFC XXXX

Applications that use this media type:

The JSON Resource Descriptor (JRD) is used by the WebFinger protocol (RFC XXXX) to enable the exchange of information between a client and a WebFinger resource over HTTPS.

Fragment identifier considerations:

The syntax and semantics of fragment identifiers SHOULD be as specified for "application/json". (At publication of this

document, there is no fragment identification syntax defined for "application/json".)

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): jrd

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Paul E. Jones <paulej@packetizer.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Paul E. Jones <paulej@packetizer.com>

Change controller:

IESG has change control over this registration.

Provisional registration? (standards tree only): N/A

10. Acknowledgments

This document has benefited from extensive discussion and review of many of the members of the APPSAWG working group. The authors would like to especially acknowledge the invaluable input of Eran Hammer-Lahav, Blaine Cook, Brad Fitzpatrick, Laurent-Walter Goix, Joe Clarke, Michael B. Jones, Peter Saint-Andre, Dick Hardt, Tim Bray, James Snell, Melvin Carvalho, Evan Prodromou, Mark Nottingham, Barry Leiba, Elf Pavlik, Bjoern Hoehrmann, SM, Joe Gregorio and others that we have undoubtedly, but inadvertently, missed. Special thanks go to the chairs of APPSAWG, especially Salvatore Loreto for his assistance in shepherding this document.

11. References

11.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,

Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[3] Nottingham, M., Hammer-Lahav, E., "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[4] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[5] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[6] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[7] Duerst, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[8] Saint-Andre, P., "The 'acct' URI Scheme", draft-ietf-appsawg-acct-uri-03, February 2013.

[9] Van Kesteren, A., "Cross-Origin Resource Sharing", W3C CORS http://www.w3.org/TR/cors/, July 2010.

[10] IANA, "Link Relations", http://www.iana.org/assignments/link-relations/.

[11] IANA, "MIME Media Types", http://www.iana.org/assignments/media-types/index.html.

[12] Freed, N., Klensin, J., Hansen, T., "Media Type Specifications and Registration Procedures", RFC 6838, January 2013.

[13] Phillips, A., Davis, M., "Tags for Identifying Languages", RFC 5646, January 2009.

[14] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[15] Klyne, G., Newman, C., "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

11.2. Informative References

[16] Perreault, S., "vCard Format Specification", RFC 6350, August 2011.

[17] "Transport Independent, Printer/System Interface", IEEE Std 1284.1-1997, 1997.

[18] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Messages 1.0", January 2013, http://openid.net/specs/openid-connect-messages-

1_0.html.

[19] Hammer-Lahav, E. and Cook, B., "Web Host Metadata", RFC 6415, October 2011.

[20] Hammer-Lahav, E. and W. Norris, "Extensible Resource Descriptor (XRD) Version 1.0", http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.html.

[21] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, October 2010.

[22] Balduzzi, Marco, et al., "Abusing social networks for automated user profiling", Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2010, https://www.eurecom.fr/en/publication/3042/download/rs-publi-3042_1.pdf.

Author's Addresses

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 476 2048
Email: paulej@packetizer.com
IM: xmpp:paulej@packetizer.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 392 3266
Email: gsalguei@cisco.com
IM: xmpp:gsalguei@cisco.com

Joseph Smarr
Google

Email: jsmarr@google.com