Network Working Group
Internet Draft
Intended status: Standards Track
Expires: January 3, 2013

Paul E. Jones
Gonzalo Salgueiro
Cisco Systems
Joseph Smarr
Google

July 3, 2012

WebFinger
draft-ietf-appsawg-webfinger-00.txt

Abstract

This specification defines the WebFinger protocol. WebFinger may be used to discover information about people on the Internet, such as a person's personal profile address, identity service, telephone number, or preferred avatar. WebFinger may also be used to learn information about objects on the network, such as the amount of toner in a printer or the physical location of a server.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

There is a utility found on UNIX systems called "finger" [14] that allows a person to access information about another person. The information being queried might be on a computer anywhere in the world. The information returned via "finger" is simply a plain text file that contains unstructured information provided by the queried user.

WebFinger borrows the concept of the legacy finger protocol, but introduces a very different approach to sharing information. Rather than returning a simple unstructured text file, Webfinger uses structured documents that contain link relations. These link relations point to information a user or entity on the Internet wishes to expose. For a person, the kinds of information that might be exposed include a personal profile address, identity service, telephone number, or preferred avatar. WebFinger may also be used to learn information about objects on the network, such as the amount of toner in a printer or the physical location of a server.

Information returned via WebFinger might be for direct human consumption (e.g., another user'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 identification service).

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". Briefly, a Link Relation is an attribute and value pair used on the Internet wherein the attribute identifies the type of link to which the associated value refers. In Hypertext Transfer Protocol (HTTP) [2] and Web Linking Error! Reference source not found., the attribute is a "rel" and the value is an "href".

3. Overview

WebFinger enables the discovery of information about accounts, devices, and other entities that are associated with web-accessible domains. In essence, there are two steps to discovering such information:

1. By querying the domain itself, one can find out how to discover information about accounts, devices, and other associated with that domain.
2. By then querying an entity at the domain, one will find links to more detailed information, which can then be queried individually.

To enable such functionality, WebFinger makes heavy use of well-known URIs as defined in RFC 5785 [3] and "Link Relations" as defined in RFC 5988 [3]. Briefly, a link is a typed connection between two web resources that are identified by Internationalized Resource Identifiers (IRIs) [13]; this connection consists of a context IRI, a link relation type, a target IRI, and optionally some target attributes, resulting in statements of the form "{context IRI} has a {relation type} resource at {target IRI}, which has {target attributes}". When used in the Link HTTP header, the context IRI is the IRI of the requested resource, the relation type is the value of the "rel" parameter, the target IRI is URI-Reference contained in the Link header, and the target attributes are the parameters such as

"hreflang", "media", "title", "title*", "type", and any other link-extension parameters.

Thus the framework for WebFinger consists of several building blocks:

1. To query the domain, one requests a web host metadata file [11] located at a well-known URI of /.well-known/host-meta at the domain of interest.
2. The web server at the domain returns an Extensible Resource Descriptor (XRD) or a JavaScript Object Notation (JSON) Resource Descriptor (JRD) document, including a Link-based Resource Descriptor Document (LRDD) link relation.
3. To discover information about accounts, devices, or other entities associated with the domain, one requests the actual Link-based Resource Descriptor Document associated with a particular URI at the domain (e.g., an 'acct' URI, 'http' URI', or 'mailto' URI).
4. The web server at the domain returns an XRD or JRD document about the requested URI, which includes specialized link relations pointing to resources that contain more detailed information about the entity.

This model is illustrated in the examples under Section 4, then described more formally under Section 5. Note that steps 2 and 3 above may be accomplished simultaneously by utilizing the "resource" parameter defined in Section 5.2.

4. Example Uses of WebFinger

In this section, we describe just a few sample uses for WebFinger and show what the protocol looks like. This is not an exhaustive list of possible uses and the entire section should be considered non-normative. The list of potential use cases is virtually unlimited since a user can share any kind of machine-consumable information via WebFinger.

4.1. 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 makes that information available via WebFinger.

Let's assume your email client discovers that blog automatically for you. After receiving the message from Bob (bob@example.com), your email client performs the following steps behind the scenes.

First, it tries to get the host metadata [11] information for the domain example.com. It does this by issuing the following HTTPS query to example.com:

```
GET /.well-known/host-meta HTTP/1.1
Host: example.com
```

The server replies with an XRD [10] document:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/xrd+xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
  <Link rel="lrdd"
        type="application/xrd+xml"
        template="https://example.com/lrdd/?uri={uri}"/>
</XRD>
```

The client then processes the received XRD in accordance with the Web Host Metadata [11] procedures. The client will see the LRDD link relation and issue a query with the user's account URI [6] or other URI that serves as an alias for the account. (The account URI is discussed in Section 4.2.) The query might look like this:

```
GET /lrdd/?uri=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/xrd+xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="http://docs.oasis-open.org/ns/xri/xrd-1.0">
  <Expires>2012-03-13T20:56:11Z</Expires>
  <Subject>acct:bob@example.com</Subject>
  <Alias>http://www.example.com/~bob/</Alias>
  <Link rel="http://webfinger.net/rel/avatar"
        href="http://www.example.com/~bob/bob.jpg"/>
  <Link rel="http://webfinger.net/rel/profile-page"
        href="http://www.example.com/~bob/"/>
  <Link rel="http://packetizer.com/rel/blog"
        href="http://blogs.example.com/bob/"/>
</XRD>
```

The email client might take note of the "blog" link relation in the above XRD document that refers to Bob's blog. This URL 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.

Note in the above example that an alias is provided that can also be used to return information about the user's account. Had the "http:" URI been used to query for information about Bob, the query would have appeared as:

```
  GET /lrdd/?uri= http%3A%2F%2Fwww.example.com%2F~bob%2F HTTP/1.1
  Host: example.com
```

The response would have been substantially the same, with the subject and alias information changed as necessary. Other information, such as the expiration time might also change, but the set of link relations and properties would be the same with either response. Let's assume, though, that for the above query the client requested a JRD representation for the resource rather than an XRD representation. In that case, the response would have been:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
    "expires" : "2012-03-13T20:56:11Z",
    "subject" : "http://www.example.com/~bob/",
    "aliases" :
    [
      "acct:bob@example.com"
    ],
    "links" :
    [
      {
        "rel" : "http://webfinger.net/rel/avatar",
        "href" : "http://www.example.com/~bob/bob.jpg"
      },
      {
        "rel" : "http://webfinger.net/rel/profile-page",
        "href" : "http://www.example.com/~bob/"
      },
      {
        "rel" : "http://packetizer.com/rel/blog",
        "href" : "http://blogs.example.com/bob/"
      }
    ]
  }
```

4.2. Retrieving a Person's Contact Information

Assume you have Alice in your address book, but her phone number appears to be invalid. You could use WebFinger to find her current phone number and update your address book.

Let's assume you have a web-based address book that you wish to update. When you instruct the address book to pull Alice's current contact information, the address book might issue a query like this to get host metadata information for example.com:

```
  GET /.well-known/host-meta.json HTTP/1.1
  Host: example.com
```

Note the address book is looking for a JSON [5] representation, whereas we used XML in the previous example.

The server might reply with something like this:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
    "links" :
    [
      {
        "rel" : "lrdd",
        "type" : "application/json",
        "template" :
          "https://example.com/lrdd/?format=json&uri={uri}"
      }
    ]
  }
```

The client processes the response as described in RFC 6415 [11]. It will process the LRDD link relation using Alice's account URI by issuing this query:

```
  GET /lrdd/?format=json&uri=acct%3Aalice%40example.com HTTP/1.1
  Host: example.com
```

The server might return a response like this:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
```

```
       "expires" : "2012-03-13T20:56:11Z",
       "subject" : "acct:alice@example.com",
       "links" :
       [
         {
           "rel" : "http://webfinger.net/rel/avatar",
           "href" : "http://example.com/~alice/alice.jpg"
         },
         {
           "rel" : "vcard",
           "href" : "http://example.com/~alice/alice.vcf"
         }
       ]
     }
```

With this response, the address book might see the vcard [16] link relation and use that file to offer you updated contact information.

4.3. Simplifying the Login Process

OpenID (http://www.openid.net) is great for allowing users to log into a web site, though one criticism is that it is challenging for users to remember the URI they are assigned. WebFinger can help address this issue by allowing users to use user@domain-style addresses. Using a user's account URI, a web site can perform a query to discover the associated OpenID identifier for a user.

Let's assume Carol is trying to use OpenID to log into a blog. The blog server might issue the following query to get the host metadata information:

```
     GET /.well-known/host-meta.json HTTP/1.1
     Host: example.com
```

The response that comes back is similar to the previous example:

```
     HTTP/1.1 200 OK
     Access-Control-Allow-Origin: *
     Content-Type: application/json; charset=UTF-8
     {
       "expires" : "2012-03-13T20:56:11Z",
       "links" :
       [
         {
           "rel" : "lrdd",
           "type" : "application/json",
           "template" :
             "https://example.com/lrdd/?format=json&uri={uri}"
         }
```

```
     ]
   }
```

The blog server processes the response as described in RFC 6415. It will process the LRDD link relation using Carol's account URI by issuing this query:

```
  GET /lrdd/?format=json&uri=acct%3Acarol%40example.com HTTP/1.1
```

The server might return a response like this:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
    "subject" : "acct:carol@example.com",
    "links" :
    [
      {
        "rel" : "http://webfinger.net/rel/avatar",
        "href" : "http://example.com/~alice/alice.jpg"
      },
      {
        "rel" : "http://specs.openid.net/auth/2.0/provider",
        "href" : "https://openid.example.com/carol"
      }
    ]
  }
```

At this point, the blog server knows that Carol's OpenID identifier is https://openid.example.com/carol and could then proceed with the login process as usual.

4.4. Retrieving Device Information

While the examples thus far have been focused on information about humans, WebFinger does not limit queries to only those that use the account URI scheme. Any URI scheme that contains domain information MAY be used with WebFinger. Let's suppose there are devices on the network like printers and you would like to check the current toner level for a particular printer identified via the URI like device:p1.example.com. 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 /lrdd/?format=json&uri=device%3Ap1.example.com HTTP/1.1
     Host: example.com
```

The link relations that are returned 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/json; charset=UTF-8

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

While this example is entirely fictitious, you can imagine that perhaps the Transport Independent, Printer/System Interface [18] may be enhanced with a web interface that allows a device that understands the TIP/SI web interface specification to query the printer for toner levels.

5. WebFinger Protocol

WebFinger does not actually introduce a new protocol, per se. Rather, it builds upon the existing Web Host Metadata [11] specification and leverages the Cross-Origin Resource Sharing (CORS) [9] specification.

5.1. Performing a WebFinger Query

The first step a client must perform in executing a WebFinger query is to query for the host metadata using HTTPS or HTTP. The procedures are defined in the Web Host Metadata [11] specification.

WebFinger clients MUST locate the LRDD link relation, if present, and perform a query for that link relation, if present. All other link templates found must be processed to form a complete resource descriptor. The processing rules in Section 4.2 of RFC 6415 MUST be followed.

WebFinger servers MUST accept requests for both XRD [10] and JRD [11] documents. The default representation returned by the server MUST be an XRD document, but a JRD document MUST be returned if the client

explicitly requests it by using /.well-known/host-meta.json or includes an Accept header in the HTTP request with a type of "application/json" [5].

If the client requests a JRD document when querying for host metadata, the WebFinger server can assume that the client will want a JRD documents when querying the LRDD resource. As such, when the WebFinger server returns a JRD document containing host metadata it should include a URI for an LRDD resource that can return a JRD document and MAY include a URI for an LRDD resource that will return an XRD document.

If the client queries the LRDD resource and provides a URI for which the server has no information, the server MUST return a 404 status code. Likewise, any query to a URI in the resource descriptor that is unknown to the server MUST result in the server returning a 404 status code.

WebFinger servers MAY include cache validators in a response to enable conditional requests by clients and/or expiration times as per RFC 2616 section 13.

5.2. The Web Host Metadata "resource" Parameter

In addition to the normal processing logic for processing host metadata information, WebFinger defines the "resource" parameter for querying for host metadata and returning all of the link relations from LRDD and other resource-specific link templates in a single query. This resource essentially pushes the work to the server to form a complete resource descriptor for the specified resource.

WebFinger servers compliant with this specification MUST support for the "resource" parameter as a means of improving performance and reducing client complexity. Note that an RFC 6415-compliant server might not implement the "resource" parameter, though the server would respond to queries from the client as described in RFC 6415. Thus, WebFinger clients MUST check the server response to ensure that the "resource" parameter is supported as explained below.

To utilize the host-meta "resource" parameter, a WebFinger client issues a request to /.well-known/host-meta or /.well-known/host-meta.json as usual, but then appends a "resource" parameter as shown in this example:

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

Note that the "\" character shown above is to indicate that the line breaks at this point and continues on the next line. This was shown only to avoid line wrapping in this document and is not a part of the HTTP protocol.

When processing this request, the WebFinger server MUST

* Return a 404 status code if the URI provided in the resource parameter is unknown to the server; and

* Set the "Subject" returned in the response to the value of the "resource" parameter if the URI provided in the resource parameter is known to the server

The WebFinger client can verify support for the "resource" parameter by checking the value of the Subject returned in the response. If the Subject matches the value of the "resource" parameter, then the "resource" parameter is supported by the server.

For illustrative purposes, the following is an example usage of the "resource" parameter that aligns with the example in Section 1.1.1 of RFC 6415. The WebFinger client would issue this request:

```
  GET /.well-known/host-meta.json?resource=\
                         http%3A%2F%2Fexample.com%2Fxy HTTP/1.1
  Host: example.com
```

The WebFinger server would reply with this response:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
    "subject" : "http://example.com/xy",
    "properties" :
    {
      "http://spec.example.net/color" : "red"
    },
    "links" :
    [
      {
        "rel" : "hub",
        "href" : "http://example.com/hub"
      },
      {
        "rel" : "hub",
        "href" : "http://example.com/another/hub"
      },
```

```
         {
           "rel" : "author",
           "href" : "http://example.com/john"
         },
         {
           "rel" : "author",
           "href" : "http://example.com/author?\
                             q=http%3A%2F%2Fexample.com%2Fxy"
         }
       ]
     }
```

5.3. The Web Host Metadata "rel" Parameter

WebFinger also defines the "rel" parameter for use when querying for host metadata. It is used to return a subset of the information that would otherwise be returned without the "rel" parameter. When the "rel" parameter is used, only the link relations that match the space-separated list of link relations provided via "rel" are included in the list of links returned in the resource descriptor. All other information normally present in a resource descriptor is present in the resource descriptor, even when "rel" is employed.

The purpose of the "rel" parameter is to return a subset of resource's link relations. It is not intended to reduce the work required of a server to produce a response. That said, 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.

Support for the "rel" parameter is OPTIONAL, but support is RECOMMENDED for both the host-meta resource and the LRDD resource.

For illustrative purposes, the following is an example usage of the "rel" parameter that aligns with the example in Section 1.1.1 of RFC 6415. The WebFinger client would issue this request to receive links that are of the type "hub" and "copyright":

```
     GET /.well-known/host-meta.json?resource=\
           http%3A%2F%2Fexample.com%2Fxy&rel=hub%20copyright HTTP/1.1
     Host: example.com
```

The WebFinger server would reply with this response:

```
     HTTP/1.1 200 OK
     Access-Control-Allow-Origin: *
     Content-Type: application/json; charset=UTF-8
```

```
     {
       "subject" : "http://example.com/xy",
       "properties" :
       {
         "http://spec.example.net/color" : "red"
       },
       "links" :
       [
         {
           "rel" : "hub",
           "href" : "http://example.com/hub"
         },
         {
           "rel" : "hub",
           "href" : "http://example.com/another/hub"
         }
       ]
     }
```

Note that in this example, the "author" links are removed, though all other content is present. Since there were no "copyright" links, none are returned.

In the event that a client requests links for link relations that are not defined for the specified resource, a resource descriptor MUST be returned, void of any links. When a JRD is returned, the "links" array MAY be either absent or empty. The server MUST NOT return a 404 status code when a particular link relation specified via "rel" is not defined for the resource, as a 404 status code is reserved for indicating that the resource itself (e.g., as indicated via the "resource" parameter) does not exist.

5.4. WebFinger and URIs

Requests for both LRDD documents and hostmeta files can include a parameter specifying the URI of an account, device, or other entity (for LRDD this is the "uri" parameter as defined by the operative XRD or JRD template, for hostmeta this is the "resource" parameter). WebFinger itself is agnostic regarding the scheme of such a URI: it could be an "acct" URI as defined in the next section, an "http" or "https" URI, a "mailto" URI, or some other scheme.

For resources associated with a user account at a domain, use of the "acct" URI scheme [7] is RECOMMENDED, since it explicitly identifies an account accessible via WebFinger. Further, the "acct" URI scheme is not associated other protocols as, by way of example, the "mailto" URI scheme is associated with email. Since not every domain offers email service, using the "mailto" URI scheme [8] is not ideal for identifying user accounts across all domains. That said, use of the "mailto" URI scheme would be ideal for use with WebFinger to discover mail server configuration information for a user, for example.

A domain MAY utilize one or more URIs that serve as aliases for the user's account, such as URIs that use the "http" URI scheme [2]. A WebFinger server MUST return substantially the same response to both an "acct" URI and any alias URI for the account, including the same set of link relations and properties. In addition, the server SHOULD include the entire list aliases for the user's account in the XRD or JRD.

6. The "acct" Link Relation

6.1. Purpose for the "acct" Link Relation

Users of some services might have an "acct" URI that looks significantly different from his or her email address, perhaps using an entirely different domain name. It is also possible for a user to have multiple accounts that a user wants to have cross-referenced from another account. To address both of these needs, this specification defines the "acct" link relation.

The "acct" link relation allows a WebFinger server to reference one or more other user account URIs from within a user account. The "acct" link relation is intended to allow a client to incorporate additional link relations by reference to produce a complete set of link relations for a user. Any newly discovered link relations found by querying the referenced account SHOULD be merged into the resource descriptor document at the point where the "acct" link relation was inserted.

Note that the "acct" link relation does not replace the use of standard HTTP 3xx response codes to indicate the new temporary or permanent location of a user account. If a user account is moved to a different location, then a 3xx response code SHOULD be used.

Since an account may make a reference to one or more different accounts, WebFinger clients MUST take steps to avoid loops wherein two account URIs, directly or indirectly, refer the client to each other.

There are no limits on the number of "acct" link relations that might be returned in a WebFinger query.

An "acct" link relation used within the context of a WebFinger query for a user's account MUST NOT return "acct" link relations for another user.

6.2. Example Message Exchange Using the "acct" Link Relation

Consider the following non-normative example.

Suppose Alice receives an email from bob@example.net. While Bob's email identifier might be in the example.net domain, he holds a user account in the example.com domain and another account in the example.org domain. His email provider may provide WebFinger services to enable redirecting Alice when she queries for acct:bob@example.net.

Suppose Alice's client issues the following request:

```
  GET /.well-known/host-meta.json?resource=\
                         acct%3Abob%40example.net HTTP/1.1
  Host: example.net
```

The response that Alice's client receives back might be:

```
  HTTP/1.1 200 OK
  Access-Control-Allow-Origin: *
  Content-Type: application/json; charset=UTF-8

  {
    "subject" : "acct:bob@example.net",
    "links" :
    [
      {
        "rel" : "acct",
        "href" : "acct:bob@example.com"
      },
      {
        "rel" : "acct",
        "href" : "acct:bob@example.org"
      },

      {
        "rel" : "acct",
        "href" : "mailto:bob@example.net"
      }
    ]
  }
```

Alice's WebFinger client could then perform queries against the URIs acct:bob@example.com, acct:bob@example.org, and mailto:bob@example.net in order to get the information Alice is seeking.

7. Cross-Origin Resource Sharing (CORS)

WebFinger is most useful when it is accessible without restrictions on the Internet, and that includes web browsers. Therefore, WebFinger servers MUST support Cross-Origin Resource Sharing (CORS) [9] when serving content intended for public consumption. Specifically, all queries to /.well-known/host-meta, /.well-known/host-meta.json, and to the LRDD URI must include the following HTTP header in the response:

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

Enterprise WebFinger servers that wish to restrict access to information from external entities SHOULD use a more restrictive Access-Control-Allow-Origin header and MAY exclude the header entirely.

8. Controlling Access to Information

As with all web resources, access to the Host Metadata resource and the LRDD resource MAY require authentication. Further, failure to provide required credentials MAY result in the server forbidding access or providing a different response than had the client authenticated with the server.

Likewise, a server 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 server response MAY point to web resources that impose access restrictions. For example, it is possible that the aforementioned corporate server may provide both internal and external entities with URIs to employee pictures, but further authentication MAY be required in order for the WebFinger client to access those resources if the request comes from outside the corporate network.

The decisions made with respect to what set of link relations a WebFinger server 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.

9. Implementation Notes (Non-Normative)

A user should not be required to enter the "acct" URI scheme name along with his account identifier into any WebFinger client. Rather, the WebFinger client should accept identifiers that are void of the "acct:" portion of the identifier. Composing a properly formatted "acct" URI is the responsibility of the WebFinger client.

10. Security Considerations

All of the security considerations applicable to Web Host Metadata [11] and Cross-Origin Resource Sharing [9] are also applicable to this specification. Of particular importance is the recommended use of HTTPS to ensure that information is not modified during transit. Clients should verify that the certificate used on an HTTPS connection is valid.

When using HTTP to request an XRD document, WebFinger clients SHOULD verify the XRD document's signature, if present, to ensure that the XRD document has not been modified. Additionally, WebFinger servers SHOULD include a signature for XRD documents served over HTTP.

Service providers and users should be aware that placing information on the Internet accessible through WebFinger means that any user can access that information. While WebFinger can be an extremely useful tool for allowing quick and easy access to one's avatar, blog, or other personal information, users should understand the risks, too. If one does not wish to share certain information with the world, do not allow that information to be freely accessible through WebFinger.

The aforementioned word of caution is perhaps worth emphasizing again with respect to dynamic information one might wish to share, such as the current location of a user. WebFinger can be a powerful tool used to assemble information about a person all in one place, 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.

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 a WebFinger server for use inside a corporate network, the network administrator must take measures necessary 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, WebFinger servers can be employed in

such a way as to require authentication and prevent disclosure of information to unauthorized entities.

11. IANA Considerations

RFC Editor: Please replace QQQQ in the following two sub-sections with a reference to this RFC.

11.1. Registration of the "acct" Link Relation Type

Relation Name: acct

Description: A link relation that refers to a user's WebFinger account identifier.

Reference: RFC QQQQ

Notes:

Application Data:

12. Acknowledgments

The authors would like to acknowledge Eran Hammer-Lahav, Blaine Cook, Brad Fitzpatrick, Laurent-Walter Goix, Joe Clarke, Mike Jones, and Peter Saint-Andre for their invaluable input.

13. References

13.1. 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] Saint-Andre, P., "The 'acct' URI Scheme", draft-saintandre-acct-uri-01, July 2012.

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

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

[10] 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.

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

[12] American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

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

13.2. Informative References

[14] Zimmerman, D., "The Finger User Information Protocol", RFC 1288, December 1991.

[15] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

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

[17] Internet Assigned Numbers Authority (IANA) Registry, "Uniform Resource Identifier (URI) Schemes", <http://www.iana.org/assignments/uri-schemes.html>.

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

[19] Hoffman, P., Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

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