

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2013

A. McMillan
Morphoss
C. Daboo
Apple Inc.
June 28, 2013

Automated Service Configuration
draft-daboo-aggregated-service-discovery-03

Abstract

This specification describes how clients can retrieve configuration for multiple services with a minimum of user-provided information, and as short as possible sequence of queries, and with a minimum of overhead for administrators of the services.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 30, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Open Issues	4
3. Conventions Used in This Document	4
4. Overview	4
5. Automated Service Configuration Document Format	4
5.1. Extensions to the Document Format	8
5.2. Service names	8
6. Finding the Automated Service Configuration Information	8
7. Handling multiple, alternative services	9
8. Internationalization Considerations	10
9. Security Considerations	10
10. IANA Considerations	10
10.1. Link Relation Registration	10
10.1.1. service-configuration Link Relation Registration	10
11. Acknowledgments	11
12. References	11
12.1. Normative References	11
12.2. Informative References	12
Appendix A. Change History (to be removed prior to publication as an RFC)	12
Appendix B. Example of multiple services	13
Appendix C. Example - multiple, alternative mail retrieval services	15
Appendix D. Example - multiple links	18

1. Introduction

There are currently various systems in place for discovery and configuration of individual services, but the process can often require an extensive series of requests using different protocols to discover all of the details needed to set up the various client services which an individual might use to interact with an organisation or service provider.

Consider Jason, a new employee at Example Enterprises. Jason needs to configure his e-mail program to use IMAP [RFC3501] + TLS on port 143 against mail.example.com, he needs to send mail on port 8557 via TLS+SMTP to smtp.example.com, his calendar is on port 8443 at https://caldav.example.com:8443/calendar/, and so forth. Some of these things can be discovered relatively easily, with a combination of DNS queries (including SRV lookups, certificate checking, and http requests). However, each protocol has its own requirements and settings and each has to be done separately. Whilst the client can "hide" the multiple service setup from the user, the actual implementation often requires separate code and processes to manage, making it more complex than it needs to be.

This specification defines a single protocol which will allow for retrieval of a variety of service configuration information in a single call, allowing developers to simplify the coding and user interface in client software, and in particular in multi-function client software such as a combined e-mail and calendar clients. Configuration is retrieved from a single document on a server, improving performance over per-service configuration mechanisms that require multiple network operations. In addition, complex dependencies between different services can be easily represented, so that, for example, some services can be prioritized over others, or grouped together by "logical" function. Further, rich information about each service can be included, such as details about required transport layer security or authentication.

This protocol is intended to provide a simple one-way retrieval of configuration information, with the data flow from service provider to client, building on successful technologies such as HTTP and JSON to accomplish its goals. As such it differs from other "client configuration" orientated protocols, such as ACAP [RFC2244], which have not had much deployment success due to complexity introduced by having a two-way exchange of data, and the resulting need to keep multiple clients synchronized as changes occur.

2. Open Issues

1. Is it OK to embed certificate details for the actual services or a root certificate?

3. Conventions Used in This Document

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 [RFC2119].

4. Overview

The following outlines the steps a client carries out to setup multiple services for a user:

1. The client software is expected to capture a user identifier and domain name (possibly entered in the form of an email address) from the user, and possibly authentication information. e.g., 'cyrus@example.com'.
2. The client uses the WebFinger [I-D.ietf-appsawg-webfinger] protocol to request a link relation with the value "service-configuration", using the identifier supplied by the user, following any redirects and responding to any authentication challenges.
3. The client retrieves a JSON [RFC4627] document conforming to the format described in Section 5 from the target of the link relation returned in Step #2. The client parses this document to extract information about the available services. At that point it can either present a list of services to the user, so that they can decide exactly what they want setup, or it can automatically setup services for all those it supports.

5. Automated Service Configuration Document Format

The automated service configuration document is an JSON [RFC4627] document. The document contains a single object with two members representing two pieces of information: overall service provider information (e.g., name, icon "badge", contact information), and a list of each service supported. Each service will contain some information common to each type of service, and then information specific to each service.

The JSON document format is defined here using the syntax in [I-D.newton-json-content-rules]. An example of such an automated service configuration document for some common services is shown in

Appendix B.

JSON Content Rules for the JSON document returned for a "capabilities" action request.

```
; root object

root {
  provider,
  entries
}

; ----- provider -----

; Contains information describing the service provider, that can be
; used by clients to "group" individual services together under a
; common name or section when presenting details to the user.
provider "provider" {
  provider_name,
  ?description,
  ?image,
  ?contact,
  ?manage,
  ?password_reset,
  ?ttl
}

; The name for the service provider.
provider_name "name" : string

; The description of the service provider.
description "description" : string

; A URI for an image that can be used as an "icon" for the service
; provider. The URI SHOULD be an http or https URI and clients
; SHOULD use standard HTTP Accept header behavior to request an
; appropriate image format from the server (see Section 14.1 of
; [RFC2616]). The image SHOULD NOT exceed a size of 128 x 128
; pixels.
image "image" : uri

; Contact information for the service provider.
contact "contact" {
  email / (?email, uri)
}

; An email address that can be used to contact the service
; provider.
```

```
email "email" : email

; A URI for a webpage providing information about the service
; provider.
url "url" : uri

; A URI for a webpage where a user can manage details of their
; account.
; e.g., a place where users can go to add additional (possibly
; payment required) services.
manage "manage" : uri

; A URI for a webpage where a user can change their account
; password.
password_reset "password-reset" : uri

; The minimum interval in seconds which clients SHOULD wait
; before re-fetching the document to check for changes.
ttl "ttl" : integer

; ----- entries -----

; List of services.
entries "entries" [ *entry ]

entry {
    name,
    service,
    ?(group, priority),
    uri / (host, ?port),
    ?tls,
    ?auth,
}

; A description for the service.
name "name" : string

; The service type. See below for details of the value used.
service "service" : string

; Identifies the nature of the service to allow similar services to
; be grouped together.
group "group" : string

; Identifies the nature of the service to allow similar services to
; be grouped together.
priority "priority" : integer 1
```

```
; The URI used to contact the server providing the service.
uri "uri" : uri

; The hostname of the server providing the service.
host "host" : string

; The network port number of the server providing the service.
port "port" : integer 0..65535

; Provides detail of transport layer security to be used with the
; service.
tls "tls" {
    ?required,
    ?at_start,
    ?certificates
}

; Indicates that clients MUST use transport layer security when
; connecting to the server providing the service.
required "required" : boolean

; Indicates that clients MUST initiate TLS immediately upon
; connecting to the server rather than using an "in-protocol"
; upgrade mechanism.
at_start "at-start" : boolean

; List of certificates.
certificates "certificates" [ *certificate ]

; Details about the TLS certificate the server will use. Clients
; MAY use the specified certificate information to validate any TLS
; connection to the server, otherwise existing rules for the target
; protocol are used.
certificate "certificate" {
    cert_name /
    fingerprint /
    public_key
}

; The name of the certificate.
cert_name "name" : string

; The fingerprint of the certificate.
fingerprint "fingerprint" : string

; The fingerprint of the certificate.
public_key "public-key" : string
```

```
; List of authentication methods to use in server preferred order.  
; If the protocol supports SASL [RFC4422] then this is a list of  
; SASL authentication mechanisms, otherwise it is a protocol  
; specific list of names. In either case, clients MUST NOT use  
; a mechanism that is not advertised in this list.  
auth "auth" [ *string ]
```

5.1. Extensions to the Document Format

Additional members can be added to the JSON document root, "provider" or "entries" objects with the following rules:

1. Standards based members MUST be defined in an RFC and registered with IANA. TBD - precise details of this and IANA registry setup.
2. Member names that include a prefix of the form "{...}", where the contents of the curly braces is a vendor id, are considered to be vendor specific private extensions which do not require registration. TBD nature of vendor id.

Clients SHOULD ignore all extension member elements that they are unable to process.

5.2. Service names

The "service" member in the "entry" object is used to convey an identifier for the type of service being described. This can have one of two forms:

1. An identifier from the IANA ports registry defining a service type.
2. Identifiers that include a prefix of the form "{...}", where the contents of the curly braces is a vendor id, are considered to be vendor specific private service type. TBD nature of vendor id.

6. Finding the Automated Service Configuration Information

This specification makes use of the WebFinger [I-D.ietf-appsawg-webfinger] protocol to allow a client to find a link to the automated service configuration document corresponding to an identifier supplied by a user. This specification registers the "service-configuration" link relation type for use with WebFinger, to identify the links for automated service configuration documents. When the client makes a WebFinger request, it SHOULD use the "rel" query parameter, defined in Section 4.3 of

[I-D.ietf-appsawg-webfinger], to ensure the relevant link relations are returned. The URI referenced by the links MUST be an HTTPS [RFC2818] URI, and MUST point to a resource that the client can use to retrieve the automated service configuration document for the site.

Servers can return multiple "service-configuration" links in the webfinger response. Clients SHOULD treat each link as separate, complementary services available to the user.

When requesting automated service configuration documents, clients MUST include a URI query parameter "id" set to the user identifier entered by the user. Clients MUST handle HTTP redirects on the link URI, but MUST NOT allow a redirect to an insecure URI. When responding to the request, the server MUST tailor the automated service configuration document for the user making the request and MUST require HTTP authentication by that user before returning the document.

Clients SHOULD cache the document for a period of time no less than the value of the "ttl" member in the "provider" object, or for a minimum of 24 hours if no "ttl" member is present.

7. Handling multiple, alternative services

The "group" and "priority" members of an entry provide a way for a service provider to distinguish multiple services of the same type, as well as allow the client to select the most appropriate service when several alternatives exist.

For example, consider the case of a service provider supporting two separate email retrieval services, one the "primary" account, and the other for "internal" messaging only. It is expected that clients configure accounts for both services. Each service also offers either IMAP [RFC3501] or POP3 [RFC1939] as an email retrieval protocol. In this case the automated service configuration document would contain four entry items: two describing an IMAP service and two describing a POP3 service. Each entry would contain a "group" member that groups one IMAP and one POP3 service together for each of the "primary" and "internal" account groups. Each entry would also contain a "priority" member indicating the service providers preference for clients to use either IMAP or POP3. An example of such an automated service configuration document is shown in Appendix C.

When a client retrieves and processes such a document, it would first group services based on the SD:application value. For each group, it iterates over the list of entries in the group, ordered by SD:

priority values, and configures an account for the first one it finds with an SD:service that it supports.

8. Internationalization Considerations

Some elements of the automated service configuration document can contain human readable text that clients might choose to present to a user. Clients SHOULD use the Accept-Language header behavior described in Section 14.4 of [RFC2616] to ensure the server can return a document suitable for the user's chosen language. Servers SHOULD support variations of the automated service configuration document based on language, returning the appropriate variation in response to client requests.

9. Security Considerations

When using WebFinger to discover a server hosting the automated service configuration document, a malicious attacker with access to the DNS server data, or able to get spoofed answers cached in a recursive resolver, can potentially cause clients to connect to a server hosting a bogus automated service configuration document with service data chosen by the attacker. In the absence of a secure DNS option, clients SHOULD check that the target FQDN returned in the link relation URI record matches the original service domain that was queried. If the target FQDN is not in the queried domain, clients SHOULD verify with the user that the link URI target FQDN is suitable for use before executing any connections to the host.

HTTP requests for the automated service configuration document MUST be performed via TLS. Clients MUST use the procedure outlined in Section 4.3 of [RFC6125] to verify the service.

10. IANA Considerations

10.1. Link Relation Registration

This document defines a "service-configuration" link relation type using the registration procedure and template from Section 6.2 of [RFC5988].

10.1.1. service-configuration Link Relation Registration

Relation Name: service-configuration

Description: Refers to an automated service configuration document

Reference: This RFC.

11. Acknowledgments

The authors would like to thank the following individuals for contributing their ideas and providing feedback for writing this specification: Andrew Biggs, Mike Douglass, Joe Hildebrand, Paul Jones, Markus Lanthaler, Matt Miller, Stepan Potys, and Peter Saint-Andre

The authors would also like to thank CalConnect, The Calendaring and Scheduling Consortium, for advice with this specification.

12. References

12.1. Normative References

- [I-D.ietf-appsawg-webfinger] Jones, P., Salgueiro, G., and J. Smarr, "WebFinger", draft-ietf-appsawg-webfinger-14 (work in progress), May 2013.
- [I-D.newton-json-content-rules] Newton, A., "A Language for Rules Describing JSON Content", draft-newton-json-content-rules-01 (work in progress), January 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] 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.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627,

July 2006.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

12.2. Informative References

- [RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.
- [RFC2244] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

Appendix A. Change History (to be removed prior to publication as an RFC)

Changes in -03:

1. Fix examples to properly show request/response headers.
2. Title change.
3. Added text to the Introduction to differentiate this from other protocols.
4. Switched to WebFinger instead of SRV.

Changes in -02:

1. Switched to JSON document format.

Changes in -01:

1. Renamed various elements for clarity.
2. Added an SD:manage element.
3. Added a section on handling of multiple, alternative services, together with a second appendix example.

Appendix B. Example of multiple services

First comes the WebFinger request to retrieve the appropriate link relation.

>> Request <<

```
GET /.well-known/webfinger?
  resource=acct:cyrus@example.com&
  rel=service-configuration HTTP/1.1
Host:example.com:443
Authorization: basic QmFzZTY0IGlzIGVhc3kgdG8gZGVjb2Rl
```

>> Response <<

```
HTTP/1.1 200 OK
Date: Wed, 20 Feb 2013 09:32:12 GMT
Content-Type: application/jrd+json
Content-Length: xxx
```

```
{
  "subject" : "acct:cyrus@example.com",
  "links" :
  [
    {
      "rel" : "service-configuration",
      "type" : "application/json",
      "href" : "https://example.com/service-config"
    }
  ]
}
```

Next is the retrieval of the automated service configuration document using the URI from the WebFinger link relation.

>> Request <<

```
GET /service-config?id=cyrus@example.com HTTP/1.1
Host:example.com:443
Authorization: basic QmFzZTY0IGlzIGVhc3kgdG8gZGVjb2Rl
```

>> Response <<

HTTP/1.1 200 OK

Date: Wed, 20 Feb 2013 09:32:12 GMT

Content-Type: application/json

Content-Length: xxx

```
{
  "provider" : {
    "name" : "Super-duper ISP",
    "description" : "Super-duper ISP is the home for all your data.",
    "contact" : {
      "email" : "superduper@example.com",
      "uri" : "http://www.example.com"
    },
    "manage" : "http://www.example.com/myaccount.html",
    "ttl" : 2592000
  },

  "entries" : [
    {
      "name" : "Corporate Mail",
      "service" : "imap",
      "group" : "mail-access-1",
      "priority" : 2,
      "uri" : "imap:imap.example.com",
      "tls" : {
        "required" : true
      },
      "auth" : ["CRAM-MD5"]
    },

    {
      "name" : "Corporate Mail",
      "service" : "pop3",
      "group" : "mail-access-1",
      "priority" : 1,
      "host" : "mail.example.com",
      "port" : 110,
      "tls" : {
        "required" : true
      },
      "auth" : ["CRAM-MD5"]
    },

    {
      "name" : "Corporate Mail",
      "service" : "submission",
```

```
    "host" : "mail.example.com",
    "port" : 587,
    "tls" : {
      "required" : true
    },
    "auth" : ["CRAM-MD5"]
  },
  {
    "name" : "Corporate Calendar",
    "service" : "caldav",
    "uri" : "https://calendar.example.com",
    "tls" : {
      "required" : true,
      "at-start" : true
    },
    "auth" : ["Digest"]
  },
  {
    "name" : "Corporate Contacts",
    "service" : "carddav",
    "uri" : "https://contacts.example.com",
    "tls" : {
      "required" : true,
      "at-start" : true
    },
    "auth" : ["Digest"]
  }
]
}
```

Appendix C. Example - multiple, alternative mail retrieval services

First comes the WebFinger request to retrieve the appropriate link relation.

>> Request <<

```
GET /.well-known/webfinger?
  resource=acct:cyrus@example.com&
  rel=service-configuration HTTP/1.1
Host:example.com:443
Authorization: basic QmFzZTY0IGlzIGVhc3kgdG8gZGVjb2Rl
```

>> Response <<

```
HTTP/1.1 200 OK
Date: Wed, 20 Feb 2013 09:32:12 GMT
Content-Type: application/jrd+json
Content-Length: xxx
```

```
{
  "subject" : "acct:cyrus@example.com",
  "links" :
  [
    {
      "rel" : "service-configuration",
      "type" : "application/json",
      "href" : "https://example.com/service-config"
    }
  ]
}
```

Next is the retrieval of the automated service configuration document using the URI from the WebFinger link relation. This example shows two different email services: "Primary Mail" (which has both IMAP4 and POP3 services available), and "Internal Mail" (which additionally has a private "webmail" service available). An extension member is also specified for the "webmail" service.

>> Request <<

```
GET /service-config?id=cyrus@example.com HTTP/1.1
Host:example.com:443
Authorization: basic QmFzZTY0IGlzIGVhc3kgdG8gZGVjb2Rl
```

>> Response <<

```
HTTP/1.1 200 OK
Date: Wed, 20 Feb 2013 09:32:12 GMT
Content-Type: application/json
Content-Length: xxx
```

```
{
```



```
"provider" : {
  "name" : "Mail Agregator ISP",
  "description" : "Primary and internal email services.",
  "contact" : {
    "email" : "emails@example.com",
    "uri" : "http://www.example.com"
  },
  "manage" : "http://www.example.com/myaccount.html",
  "ttl" : 2592000
},

"entries" : [
  {
    "name" : "Primary Mail",
    "service" : "imap",
    "group" : "primary",
    "priority" : 2,
    "uri" : "imap:mail.example.com",
    "tls" : {
      "required" : true
    },
    "auth" : [ "CRAM-MD5" ]
  },

  {
    "name" : "Primary Mail",
    "service" : "pop3",
    "group" : "primary",
    "priority" : 1,
    "host" : "mail.example.com",
    "port" : 110,
    "tls" : {
      "required" : true
    },
    "auth" : [ "CRAM-MD5" ]
  },

  {
    "name" : "Internal Mail",
    "service" : "imap",
    "group" : "internal",
    "priority" : 2,
    "uri" : "imap:int.example.com",
    "tls" : {
      "required" : true
    },
    "auth" : [ "CRAM-MD5" ]
  },
]
```

```
{
  "name" : "Internal Mail",
  "service" : "pop3",
  "group" : "internal",
  "priority" : 1,
  "host" : "int.example.com",
  "port" : 110,
  "tls" : {
    "required" : true
  },
  "auth" : [ "CRAM-MD5" ]
}

{
  "name" : "Internal Mail",
  "service" : "{example.com}webmail",
  "group" : "internal",
  "priority" : 1,
  "uri" : "https://int.example.com/webmail",
  "{example.com}bookmark": "https://int.example.com/webmail"
}
]
}
```

Appendix D. Example - multiple links

TODO: example of webfinger returning multiple "service-configuration" links.

Authors' Addresses

Andrew McMillan
Morphoss Ltd
6 Karoro Place
Porirua 5024
New Zealand

EMail: andrew@morphoss.com
URI: <http://www.morphoss.com/>

Cyrus Daboo
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
USA

EMail: cyrus@daboo.name
URI: <http://www.apple.com/>

Network Working Group
Internet Draft
Intended status: Standards Track
Expires: December 18, 2012

Paul E. Jones
Gonzalo Salgueiro
Cisco Systems
Joseph Smarr
Google
June 18, 2012

WebFinger
draft-jones-appsawg-webfinger-06.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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 18, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction.....	2
2. Terminology.....	3
3. Overview.....	3
4. Example Uses of WebFinger.....	4
4.1. Locating a User's Blog.....	4
4.2. Retrieving a Person's Contact Information.....	7
4.3. Simplifying the Login Process.....	8
4.4. Retrieving Device Information.....	9
5. WebFinger Protocol.....	10
5.1. Performing a WebFinger Query.....	10
5.2. The Web Host Metadata "resource" Parameter.....	11
5.3. The Web Host Metadata "rel" Parameter.....	13
5.4. WebFinger and URIs.....	14
6. The "acct" URI.....	15
7. The "acct" Link Relation.....	16
7.1. Purpose for the "acct" Link Relation.....	16
7.2. Example Message Exchange Using the "acct" Link Relation..	16
8. Cross-Origin Resource Sharing (CORS).....	17
9. Controlling Access to Information.....	17
10. Implementation Notes (Non-Normative).....	18
11. Security Considerations.....	18
12. IANA Considerations.....	19
12.1. Registration of the "acct" URI scheme name.....	19
12.2. Registration of the "acct" Link Relation Type.....	20
13. Acknowledgments.....	20
14. References.....	20
14.1. Normative References.....	20
14.2. Informative References.....	21
Author's Addresses.....	22

1. Introduction

There is a utility found on UNIX systems called "finger" [15] 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) [14]; 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 [10] 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 [10] 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 [9] 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 [10] 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 [10]. 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 [17] 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:pl.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 [19] 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 [10] specification and leverages the Cross-Origin Resource Sharing (CORS) [8] 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 [10] 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 [9] and JRD [10] 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 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 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. 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" URI

The "acct" URI takes a familiar form in looking like an email address. However, the account URI is not an email address and should not be mistaken for one. Quite often, the account URI minus the "acct:" scheme prefix may be exactly the same as the user's email address.

The "acct" URI syntax is defined here in Augmented Backus-Naur Form (ABNF) [7] and borrows syntax elements from RFC 3986 [6]:

```
acctURI      = "acct:" userpart "@" domainpart
userpart     = 1*( unreserved / pct-encoded )
domainpart   = domainlabel 1*( "." domainlabel )
domainlabel  = alphanum / alphanum *( alphanum / "-" ) alphanum
alphanum     = ALPHA / DIGIT
```

The "acct" URI scheme allows any character from the Unicode [12] character set encoded as a UTF-8 [20] string that is then percent-encoded as necessary into valid ASCII [21]. Characters in the domainpart must be encoded to support internationalized domain names (IDNs) [13].

Characters in the userpart or domainpart that are not unreserved must be percent-encoded when used in a protocol or document that only supports or requires ASCII. When carried in a document (e.g., XRD or JRD) or protocol that supports the Unicode character set (e.g., UTF-8 or UTF-16 [22]), the URI strings may appear in the protocol or document's native encoding without percent-encoding. Such usage of a URI is commonly referred to as an Internationalized Resource Identifier (IRI). Conversion between an IRI and URI is described in Section 3 of RFC 3987 [14].

7. The "acct" Link Relation

7.1. 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 have multiple accounts that a user wants to advertise and that a WebFinger client may want to query. To address both of these needs, this specification defines the "acct" link relation.

Since an account may make a reference to one or more different accounts, WebFinger clients **MUST** take steps to avoid loops wherein two accounts, 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 individual.

7.2. 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 his account with an "acct" URI in the example.com 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"
  }
]
```

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

8. 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) [8] 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.

9. 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.

10. 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.

11. Security Considerations

All of the security considerations applicable to Web Host Metadata [10] and Cross-Origin Resource Sharing [8] 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.

12. IANA Considerations

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

12.1. Registration of the "acct" URI scheme name

This specification requests IANA to register the "acct" URI scheme in the "Permanent URI Schemes" sub-registry in the "Uniform Resource Identifier (URI) Schemes" IANA registry [18]. This registration follows the URI Scheme Registration Template detailed in Section 5.4 of RFC 4395 [16].

URI scheme name: acct

Status: Permanent

URI scheme syntax: see Section 5.2 of RFC QQQQ

URI scheme semantics: see Section 5 of RFC QQQQ

Encoding considerations: The "acct" URI scheme allows any character from the Unicode character set encoded as a UTF-8 string that is then percent-encoded as necessary to result in an internal representation in US-ASCII [11]

Applications/protocols that use this URI scheme name: WebFinger

Security considerations: see Section 7 of RFC QQQQ

Contact: Gonzalo Salgueiro <gsalguei@cisco.com>

Author/Change controller: IETF <ietf@ietf.org>

References: See Section 10 of RFC QQQQ

12.2. 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:

13. 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.

14. References

14.1. 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] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [8] Van Kesteren, A., "Cross-Origin Resource Sharing", W3C CORS <http://www.w3.org/TR/cors/>, July 2010.

- [9] 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>.
- [10] Hammer-Lahav, E. and Cook, B., "Web Host Metadata", RFC 6415, October 2011.
- [11] American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.
- [12] The Unicode Consortium. The Unicode Standard, Version 6.1.0, (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-02-3) <http://www.unicode.org/versions/Unicode6.1.0/>.
- [13] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [14] Duerst, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

14.2. Informative References

- [15] Zimmerman, D., "The Finger User Information Protocol", RFC 1288, December 1991.
- [16] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [17] Perreault, S., "vCard Format Specification", RFC 6350, August 2011.
- [18] Internet Assigned Numbers Authority (IANA) Registry, "Uniform Resource Identifier (URI) Schemes", <http://www.iana.org/assignments/uri-schemes.html>.
- [19] "Transport Independent, Printer/System Interface", IEEE Std 1284.1-1997, 1997.
- [20] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.
- [21] Information Systems -- Coded Character Sets 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986, December 30, 1986.
- [22] 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

Change Log (To Be Deleted Before Publication)

=====

-06 Draft

- * Added an overview section
- * Made changes to example to show use of aliases
- * Added text to highlight that WebFinger may use various URI schemes
- * Reduced the text in the "acct" URI scheme section
- * Added an Implementation Notes section

-05 Draft

- * Minor editorial corrections
- * Removed last paragraph from Section 6.1
- * Clarified use of CORS and how it may differ for enterprise use

-04 Draft

- * Added text that makes the "resource" parameter required
- * Added a new section 8 that discusses controlling access to information
- * Added a little more to the security considerations section to briefly cover what was more fully explained in the new section 8

-03 Draft

- * Changed the name from Webfinger to WebFinger (common usage)
- * Added a new paragraph to Section 4.1 to remind readers that WebFinger benefits from all of the existing HTTP caching functionality
- * Added the "rel" parameter to allow filtering the results of a WebFinger query to include Links of the specified type(s)
- * Corrected a reference to an obsoleted RFC

- * Removed extraneous text from the terminology section

-02 Draft

- * Minor editorial changes
- * Added <Expires/> to the XML example to highlight that this element exists, since some may not be aware
- * Changed some of the link relation values, particularly for those that are not yet standardized
- * Added a note about "device:" not being standard
- * Overhauled the "acct" link relation text, breaking the normative and non-normative pieces apart
- * Added additional text to the security considerations section related to dynamic information (e.g., geographic information)

