Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 9, 2013

J. Richer, Ed.
The MITRE Corporation
T. Hardjono
MIT
M. Machulak
Newcastle University
E. Maler
XMLgrrl.com
C. Scholz
COM.lounge GmbH
N. Sakimura
NRI
J. Bradley
Ping Identity
M. Jones
Microsoft
November 5, 2012

OAuth Dynamic Client Registration Protocol
draft-ietf-oauth-dyn-reg-01

Abstract

This specification proposes an OAuth Dynamic Client Registration protocol.

Table of Contents

1. Introduction

In some use-case scenarios, it is desirable or necessary to allow OAuth clients to obtain authorization from an OAuth authorization server without the two parties having previously interacted. Nevertheless, in order for the authorization server to accurately represent to end-users which client is seeking authorization to access the end-user's resources, a method for automatic and unique registration of clients is needed. The OAuth2 authorization framework does not define how the relationship between the Client and the Authorization Server is initialized, or how a given client is assigned a unique Client Identifier. Historically, this has happened out-of-band from the OAuth protocol. This draft provides a mechanism for a client to register itself with the Authorization Server, which can be used to dynamically provision a Client Identifier, and optionally a Client Secret.

As part of the registration process, this specification also defines a mechanism for the client to present the Authorization Server with a set of meta information, such as a display name and icon to be presented to the user during the authorization step. This draft provides a method for the client to register and update this information over time.

1.1. Notational Conventions

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].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.2. Terminology

This specification uses the terms "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by OAuth 2.0 [OAuth2.0].

This specification defines the following additional terms:

- Client Registration Endpoint: The OAuth 2.0 Endpoint through which a Client can request new registration and manage the metadata associated with it.

- Registration Access Token: An OAuth 2.0 Bearer Token issued by the Authorization Server through the Client Registration Endpoint which is used by the Client to authenticate itself during update and secret rotation operations.

1.3. Requirements

[[Following are proposed requirements for dynamic client registration. This section is intended for discussion and will likely be removed in the final draft.]]

1.3.1. The client needs to be uniquely identifiable by the authorization server

In order for an authorization server to do proper user-delegated authorization and prevent unauthorized access it must be able to identify clients uniquely. As is done today in OAuth, the client identifier (and optional secret) should thus be issued by the authorization server and not simply accepted as proposed by the client.

1.3.2. The authorization server must collect metadata about a client for later user interaction

In order for the authorization server to describe a client to an end-user in an authorization step it needs information about the client. This can be the client name at a minimum, but today servers usually request at least a description, a homepage URL, and an icon when doing manual registration.

1.3.3. The authorization server should have the option of strongly authenticating the client and its metadata

In order to prevent spoofing of clients and enable dynamic building of strong trust relationships, the authorization server should have the option to verify the provided information. This might be solved using message signature verification.

1.3.4. Dynamic client registration must be possible from both web-server applications and applications with other capabilities and limitations, such as native applications

Each instance of a native application (that is, the specific instance running on each device) that is installed and run by the same user may need the option of getting a unique client identifier. In this case, there are implications around gathering and displaying enough information to ensure that the end-user is delegating authorization to the intended application. The registration protocol should be

simple and flexible enough to allow for multiple types of applications.

1.3.5. Transaction integrity must be ensured

When a client sends information to a server endpoint, it might take time for this data to propagate through big server installations that spread across various data centers. Care needs to be taken that subsequent interactions with the user after the registration process, such as an authorization request, show the correct data.

2. Client Registration Endpoint

The Client Registration Endpoint is an OAuth 2.0 Endpoint defined in this document that is designed to allow a Client to register itself with the Authorization Server. The Client Registration Endpoint MUST accept HTTP POST messages with request parameters encoded in the entity body using the "application/x-www-form-urlencoded" format. The Client Registration Endpoint MUST be protected by a transport-layer security mechanism when sending requests to the Registration Endpoint. The server MUST support TLS 1.2 RFC 5246 [RFC5246] and/or TLS 1.0 [RFC2246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the Client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125].

The Endpoint defines three operations that a client can take on it, switched by the "operation" parameter:

- client_associate: generate a new Client Identifier (and optionally a Client Secret) and associate it with the set of presented metadata (Section 3)

- client_update: update the metadata (Section 3) associated with a Client Identifier

- rotate_secret: issue a new Registration Access Token and, if applicable, a Client Secret for a Client

In order to facilitate registered clients updating their information, the Client Registration Endpoint issues a request_access_token for clients to securely identify themselves in future connections. As such, the Endpoint MUST accept requests with OAuth 2.0 Bearer Tokens [OAuth.Bearer] for these operations.

In order to support open registration and facilitate wider interoperability, the Client Registration Endpoint SHOULD allow

client_associate requests with no further authentication. These requests MAY be rate-limited to prevent a denial-of-service attack on the Client Registration Endpoint.

In addition, the Client Registration Endpoint MAY accept an initial authorization credential in the form of an OAuth 2.0 [OAuth2.0] access token in order to limit registration to only previously authorized parties. The method by which this access token is obtained by the registrant is generally out-of-band and is out of scope of this specification.

These two aspects, operation selection and client authentication, are represented by two parameters common to all operations:

operation REQUIRED. Values are "client_associate" (for new registrations), "rotate_secret" to request rotation of the "client_secret", and "client_update" (for updating parameters of an existing "client_id").

access_token OPTIONAL. An OAuth2 Bearer token used to access the Client Registration Endpoint, as defined in OAuth2 Bearer. This parameter MUST NOT be sent if the Access Token is sent in the HTTP Authorization header as described in Section 7.1 of OAuth 2.0 [OAuth2.0]. Access Tokens sent in the authorization header must be OAuth 2.0 Bearer Tokens [OAuth.Bearer].

Each operation takes a different parameter set, and all operations are described below.

The Client Registration Endpoint MUST ignore all parameters it does not understand.

2.1. Client Association Request

This operation registers a new client to the Authorization Server. The Authorization Server assigns this client a unique Client Identifier, optionally assigns a Client Secret, and associates the metadata given in the request with the issued Client Identifier. The request includes the two parameters described above as well as any parameters described in Client Metadata (Section 3).

operation REQUIRED, MUST have the value "client_associate"

access_token OPTIONAL, used to restrict new client registration

redirect_uris
REQUIRED

client_name RECOMMENDED

client_url
RECOMMENDED

logo_url OPTIONAL

contacts OPTIONAL

tos_url OPTIONAL

token_endpoint_auth_method OPTIONAL

policy_url OPTIONAL

jwk_url OPTIONAL

jwk_encryption_url OPTIONAL

x509_url OPTIONAL

x509_encryption_url OPTIONAL

require_signed_request_object OPTIONAL

default_max_age OPTIONAL

default_acr OPTIONAL

For example, a client could send the following registration request to the Client Registration Endpoint:

Following is a non-normative example request (with line wraps for display purposes only):

```
POST /register HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ... fQ.8Gj_-sj ... _X

operation=client_associate
&redirect_uris=https://client.example.org/callback
    %20https://client.example.org/callback2
&client_name=My%20Example%20
&logo_url=https://client.example.org/logo.png
&token_endpoint_auth_type=client_secret_basic
&jwk_url=https://client.example.org/my_rsa_public_key.jwk
```

2.2. Client Association Response

Upon successful association, the Client Registration Endpoint returns the newly-created Client Identifier and, optionally, a Client Secret. The response also contains a Registration Access Token that is to be used by the client to perform subsequent operations at this endpoint. These items are returned as a JSON document with the following fields as top-level members of the root JSON object.

client_id REQUIRED. The unique Client identifier, MUST NOT be currently valid for any other registered Client.

client_secret OPTIONAL. The Client secret. This MUST be unique for each "client_id". This value us used by confidential clients. It is not required for clients selecting a token_endpoint_auth_type of "private_key_jwt"

registration_access_token REQUIRED The Access token to be used by the client to perform "client_update" and "rotate_secret" requests.

issued_at
OPTIONAL. Specifies the timestamp when the identifier was issued. The timestamp value MUST be a positive integer. The value is expressed in the number of seconds since January 1, 1970 00:00:00 GMT.

expires_at OPTIONAL. The number of seconds from 1970-01-01T0:0:0Z as measured in UTC that the "client_secret" will expire or "0" if they do not expire. See RFC 3339 [RFC3339] for details regarding date/times in general and UTC in particular.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 "client_id":"s6BhdRkqt3",
 "client_secret":
  "cf136dc3c1fd9153029bb9c6cc9ecead918bad9887fce6c93f31185e5885805d",
  "registration_access_token": "this.is.a.access.token.value.ffx83",
 "expires_at":2893276800
}
```

2.3. Client Update Request

This operation updates a previously-registered client with new metadata at the Authorization Server. This request MUST be protected by the Registration Authorization Token associated with the Client Identifier. This request MAY include any fields described in Client Metadata (Section 3). The values of Client Metadata fields in this request MUST replace (not augment) the values previously associated with this client_identifier. Empty values in Client Metadata SHOULD be taken as a request to clear any existing value of that field.

operation REQUIRED, MUST have the value "client_update"

access_token REQUIRED, unless presented in the Authorization Header as in OAuth2 Bearer [OAuth.Bearer]. The Registration Access Token that was issued during the client_associate step, or previous client_update or rotate_secret calls.

redirect_uris
REQUIRED

client_name RECOMMENDED

client_url
RECOMMENDED

logo_url OPTIONAL

contacts OPTIONAL

tos_url OPTIONAL

token_endpoint_auth_method OPTIONAL

policy_url OPTIONAL

jwk_url OPTIONAL

jwk_encryption_url OPTIONAL

x509_url OPTIONAL

x509_encryption_url OPTIONAL

require_signed_request_object OPTIONAL

default_max_age OPTIONAL

default_acr OPTIONAL

For example, a client could send the following registration request to the Client Registration Endpoint:

Following is a non-normative example request (with line wraps for display purposes only):

```
POST /register HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ... fQ.8Gj_-sj ... _X

operation=client_update
&redirect_uris=https://client.example.org/callback
    %20https://client.example.org/callback2
&client_name=My%20Example%20
&logo_url=https://client.example.org/logo.png
&token_endpoint_auth_type=client_secret_basic
&jwk_url=https://client.example.org/my_rsa_public_key.jwk
```

2.4. Client Update Response

Upon successful update, the Client Registration Endpoint returns a JSON document with the following fields as top-level members of the root JSON object.

client_id REQUIRED. The unique Client identifier, MUST NOT be currently valid for any other registered Client.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 "client_id":"s6BhdRkqt3",
}
```

[[Editor's note: should this return the entire client data object, for confirmation and review, including any fields that may have been asserted by the AS?]]

2.5. Rotate Secret Request

This operation allows the client to rotate its current Client Secret, if it has one. If the client has not been issued a Client Secret, this operation is an error. [[Editor's note: could this request be used to rotate the Registration Access Token, even when there's not a client_secret? Should something else be used to rotate the token independently? This is an open issue.]]

operation REQUIRED. MUST have the value rotate_secret

access_token REQUIRED. The Registration Access Token that was issued during the client_associate step, or previous client_update or rotate_secret calls.

Following is a non-normative example request (with line wraps for display purposes only):

```
POST /register HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ... fQ.8Gj_-sj ... _X

operation=rotate_secret
```

2.6. Rotate Secret Response

Upon successful rotation of the client secret, the Client Registration Endpoint returns a JSON document with the following fields as top-level members of the root JSON object.

client_id REQUIRED. The unique Client identifier, MUST NOT be currently valid for any other registered Client.

client_secret REQUIRED. The Client secret. This MUST be unique for each "client_id".

registration_access_token REQUIRED The Access token to be used by the client to perform subsequent "client_update" and "rotate_secret" requests.

issued_at
OPTIONAL. Specifies the timestamp when the identifier was issued. The timestamp value MUST be a positive integer. The value is expressed in the number of seconds since January 1, 1970 00:00:00 GMT.

expires_at OPTIONAL. The number of seconds from 1970-01-01T0:0:0Z as measured in UTC that the "client_secret" will expire or "0" if they do not expire. See RFC 3339 [RFC3339] for details regarding date/times in general and UTC in particular.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 "client_id":"s6BhdRkqt3",
 "client_secret":
  "cf136dc3c1fd9153029bb9c6cc9ecead918bad9887fce6c93f31185e5885805d",
  "registration_access_token": "this.is.a.access.token.value.ffx83",
 "expires_at":2893276800
}
```

2.7. Client Registration Error Response

When an OAuth error condition occurs, the Client Registration Endpoint returns an Error Response as defined in Section 5.2 of the OAuth 2.0 specification.

When a registration error condition occurs, the Client Registration Endpoint returns a HTTP 400 status code including a JSON object describing the error in the response body.

The JSON object contains two members:

error The error code, a single ASCII string.

error_description The additional text description of the error for debugging.

This specification defines the following error codes:

invalid_operation The value of "operation" is invalid or not supported.

invalid_redirect_uri The value of one or more "redirect_uris" is invalid.

invalid_client_metadata The value of one of the client metadata (Section 3) fields is invalid.

Following is a non-normative example of an error response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
 "error":"invalid_operation",
 "error_description":"The value of the operation parameter must be one of
client_associate, rotate_secret or client_update."
}
```

3. Client Metadata

Clients generally have an array of metadata associated with their unique Client Identifier at the Authorization Server. These can range from human-facing display strings, such as a client name, to items that impact the security of the protocol,

Extensions and profiles of this specification MAY expand this list, but MUST at least accept all parameters on this list. The Authorization Server MUST ignore any additional parameters sent by the Client that it does not understand.

redirect_uris
REQUIRED A space-delimited list of redirect URIs.

client_name RECOMMENDED. Human-readable name of the Client to be presented to the user.

client_url
RECOMMENDED. This field contains the URL of the homepage of the client.

logo_url OPTIONAL. A URL that references a logo for the Client application. If present, the server SHOULD display this image to the end user during approval.

contacts OPTIONAL. Space delimited list of email addresses for people allowed to administer the information for this Client. This is used by some providers to enable a web UI to modify the Client information.

tos_url OPTIONAL. URL that points to a human-readable Terms of Service for the Client. The Authorization Server SHOULD display this URL to the End-User if it is given.

token_endpoint_auth_method OPTIONAL. The requested authentication type for the Token Endpoint. The options are "client_secret_post", "client_secret_basic", "client_secret_jwt", and "private_key_jwt". Other Authentication methods may be defined by extension. If unspecified or omitted, the default is "client_secret_basic" HTTP Basic Authentication Scheme as specified in Section 2.3.1 of OAuth 2.0 [OAuth2.0]. [[this list of terms needs to be expanded and fully defined, especially in reference to signed-jwt client authentication]]

policy_url OPTIONAL. A URL location that the Client provides to the End-User to read about the how the profile data will be used. The Authorization Server SHOULD display this URL to the End-User if it is given.

jwk_url OPTIONAL. URL for the Client's JSON Web Key [JWK] document that is used for signing Token Endpoint Requests. If jwk_encryption_url is not provided, the key at jwk_url is also used as the key to encrypt responses to the Client. If the Client registers both "x509_url" and "jwk_url", the keys contained in both formats MUST be the same.

jwk_encryption_url OPTIONAL. URL for the Client's JSON Web Key [JWK] that is used to encrypt any responses to the Client. If the Client registers both "jwk_encryption_url" and "x509_encryption_url", the keys contained in both formats MUST be the same.

x509_url OPTIONAL. URL for the Client's PEM encoded X.509 Certificate or Certificate chain that is used for signing Token Endpoint Requests. If "x509_encryption_url" is not provided, "x509_url" it is also used to encrypt responses to the Client. If the Client registers both "x509_url" and "jwk_url", the keys contained in both formats MUST be the same.

x509_encryption_url OPTIONAL. URL for the Client's PEM encoded X.509 Certificate or Certificate chain that is used to encrypt the ID Token and User Info Endpoint Responses to the Client. If the Client registers both "jwk_encryption_url" and "x509_encryption_url", the keys contained in both formats SHOULD be the same.

require_signed_request_object OPTIONAL. The JWS [JWS] "alg" algorithm [JWA] that MUST be required by the Authorization Server. The valid values are listed in Section 3.1 of JWA [JWA]. Servers SHOULD support "RS256".

default_max_age OPTIONAL. (default max authentication age): Type: Integer - Specifies that the End-User must be actively authenticated if any present authentication is older than the specified number of seconds. (The "max_age" request parameter corresponds to the OpenID 2.0 PAPE "max_auth_age" request parameter.) The "max_age" claim in the request object overrides this default value.

default_acr OPTIONAL. (default authentication context class reference): Type: String - Specifies the default value that the Authorization server must use for processing requests from this client. The "acrs_supported" element of discovery contains a list of the supported "acr" values for this server. The "acr" claim in the request object overrides this default value.

4. IANA Considerations

This document makes no requests of IANA.

5. Security Considerations

[[Editor's note: Following are some security considerations taken whole from the UMA and OpenID Connect source drafts.]]

- No client authentication: The server should treat unsigned pushed client metadata as self-asserted.

- Weak client authentication: The server should treat unsigned pulled client metadata as self-asserted unless the domain of the client matches the client metadata URL and the URL is well-known and trusted.

- Strong client authentication: The server should treat signed client metadata (pushed or pulled) and a signed metadata URL as

self-asserted unless it can verify the signature as being from a trusted source.

Since requests to the Client Registration Endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the server MUST require the use of a transport-layer security mechanism when sending requests to the Registration Endpoint. The server MUST support TLS 1.2 RFC 5246 [RFC5246] and/or TLS 1.0 [RFC2246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the Client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125].

Requests to the Registration Endpoint for "client_update" MUST have some rate limiting on failures to prevent the Client secret from being disclosed though repeated access attempts.

A rogue RP might use the logo for the legitimate RP, which it is trying to impersonate. An IdP needs to take steps to mitigate this phishing risk, since the logo could confuse users into thinking they're logging in to the legitimate RP. An IdP could also warn if the domain/site of the logo doesn't match the domain/site of redirect URIs. An IdP can also make warnings against untrusted RPs in all cases, especially if they're dynamically registered, have not been trusted by any users at the IdP before, and want to use the logo feature.

In a situation where the Authorization Server is supporting open Client registration, it must be extremely careful with any URL provided by the Client that will be displayed to the user (e.g. "logo_url" and "policy_url"). A rogue Client could specify a registration request with a reference to a drive-by download in the "policy_url". The Authorization Server should check to see if the "logo_url" and "policy_url" have the same host as the hosts defined in the array of "redirect_uris".

6. Acknowledgments

The authors thank the User-Managed Access Work Group and the OpenID Connect Working Group participants for their input to this document.

7. Document History

[[to be removed by RFC editor before publication as an RFC]]

- 01

- Merged UMA and OpenID Connect registrations into a single document
- Changed to form-paramter inputs to endpoint
- Removed pull-based registration

- 00

- Imported original UMA draft specification

8. References

8.1. Normative References

[JSON] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", 2006, <http://tools.ietf.org/html/rfc4627>.

[JWA] Jones, M., "JSON Web Algorithms", May 2012.

[JWE] Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", May 2012.

[JWK] Jones, M., "JSON Web Key (JWK)", May 2012.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature", May 2012.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token", May 2012.

[OAuth-Sig] Balfanz, D., "OAuth Signature proposals", 2010, <http://www.ietf.org/mail-archive/web/oauth/current/msg03893.html>.

[OAuth.Bearer] Jones, M. and D. Hardt, "OAuth 2.0 Protocol: Bearer Tokens", Aug 2012.

[OAuth2.0] Hardt, D., "OAuth 2.0 Authorization Protocol", July 2012.

[OpenID.Messages] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Messages 1.0", May 2012.

[OpenID.Session]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and N. Agarwal, "OpenID Connect Session Management 1.0", August 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 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.

[USA15] Davis, M., Whistler, K., and M. Duerst, "Unicode Normalization Forms", Unicode Standard Annex 15, 09 2009.

[hostmeta]
Hammer-Lahav, E., "Web Host Metadata", 2010, <draft-hammer-hostmeta-13.xml">http://
draft-hammer-hostmeta-13.xml">xml.resource.org/public/rfc/bibxml3/
draft-hammer-hostmeta-13.xml">reference.I-D.draft-hammer-hostmeta-13.xml>.

8.2. Non-Normative References

[UMA-Core]
Scholz, C., "UMA Requirements", 2010, <http://tools.ietf.org/id/draft-hardjono-oauth-umacore-04.txt>.

[UMA-Reqs]
Maler, E., "UMA Requirements", 2010, <http://

kantarainitiative.org/confluence/display/uma/
UMA+Requirements>.

[UMA-UC] Akram, H., "UMA Explained", 2010, <http://
kantarainitiative.org/confluence/display/uma/
UMA+Scenarios+and+Use+Cases>.

Authors' Addresses

Justin Richer (editor)
The MITRE Corporation

Phone:
Fax:
Email: jricher@mitre.org
URI:

Thomas Hardjono
MIT

Phone:
Fax:
Email: hardjono@mit.edu
URI:

Maciej Machulak
Newcastle University

Email: m.p.machulak@ncl.ac.uk
URI: http://ncl.ac.uk/

Eve Maler
XMLgrrl.com

Email: eve@xmlgrrl.com
URI: http://www.xmlgrrl.com

Christian Scholz
COM.lounge GmbH

Phone:
Fax:
Email:
URI:

Nat Sakimura
Nomura Research Institute, Ltd.

Email: n-sakimura@nri.co.jp

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Michael B. Jones
Microsoft

Email: mbj@microsoft.com