OAuth
Internet-Draft
Intended status: Standards Track
Expires: December 3, 2018

W. Denniss
Google
J. Bradley
Ping Identity
M. Jones
Microsoft
H. Tschofenig
ARM Limited

June 01, 2018

OAuth 2.0 Device Flow for Browserless and Input Constrained Devices
draft-ietf-oauth-device-flow-10

Abstract

This OAuth 2.0 authorization flow for browserless and input constrained devices, often referred to as the device flow, enables OAuth clients to request user authorization from devices that have an Internet connection, but don't have an easy input method (such as a smart TV, media console, picture frame, or printer), or lack a suitable browser for a more traditional OAuth flow. This authorization flow instructs the user to perform the authorization request on a secondary device, such as a smartphone. There is no requirement for communication between the constrained device and the user's secondary device.

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 https://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 3, 2018.

Table of Contents

1. Introduction

This OAuth 2.0 protocol flow for browserless and input constrained devices, often referred to as the device flow, enables OAuth clients to request user authorization from devices that have an internet connection, but don't have an easy input method (such as a smart TV, media console, picture frame, or printer), or lack a suitable browser for a more traditional OAuth flow. This authorization flow instructs the user to perform the authorization request on a secondary device, such as a smartphone.

The device flow is not intended to replace browser-based OAuth in native apps on capable devices (like smartphones). Those apps should follow the practices specified in OAuth 2.0 for Native Apps OAuth 2.0 for Native Apps [RFC8252].

The only requirements to use this flow are that the device is connected to the Internet, and able to make outbound HTTPS requests, be able to display or otherwise communicate a URI and code sequence to the user, and that the user has a secondary device (e.g., personal computer or smartphone) from which to process the request. There is no requirement for two-way communication between the OAuth client and the user-agent, enabling a broad range of use-cases.

Instead of interacting with the end-user's user-agent, the client instructs the end-user to use another computer or device and connect to the authorization server to approve the access request. Since the client cannot receive incoming requests, it polls the authorization server repeatedly until the end-user completes the approval process.

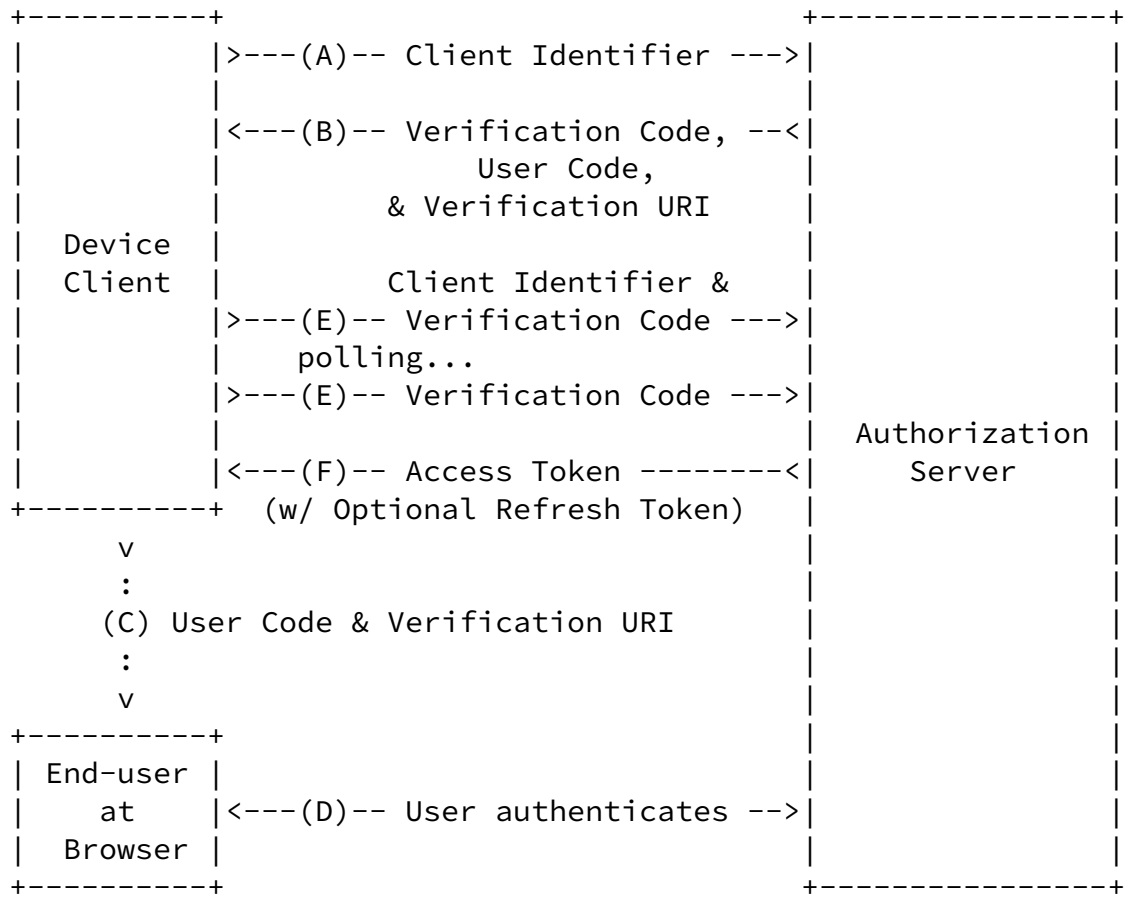

Figure 1: Device Flow.

The device flow illustrated in Figure 1 includes the following steps:

(A) The client requests access from the authorization server and includes its client identifier in the request.

(B) The authorization server issues a verification code, an end-user code, and provides the end-user verification URI.

(C) The client instructs the end-user to use its user-agent (elsewhere) and visit the provided end-user verification URI. The client provides the end-user with the end-user code to enter in order to grant access.

(D) The authorization server authenticates the end-user (via the user-agent) and prompts the end-user to grant the client's access request. If the end-user agrees to the client's access request, the end-user enters the end-user code provided by the client. The authorization server validates the end-user code provided by the end-user.

(E) While the end-user authorizes (or denies) the client's request (step D), the client repeatedly polls the authorization server to find out if the end-user completed the end-user authorization step. The client includes the verification code and its client identifier.

(F) Assuming the end-user granted access, the authorization server validates the verification code provided by the client and responds back with the access token.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Device Authorization Endpoint:
: The authorization server's endpoint capable of issuing device verification codes, user codes, and verification URLs.

Device Verification Code:

A short-lived token representing an authorization session.

End-User Verification Code:
A short-lived token which the device displays to the end user, is entered by the end-user on the authorization server, and is thus used to bind the device to the end-user.

3. Protocol

3.1. Device Authorization Request

The client initiates the flow by requesting a set of verification codes from the authorization server by making an HTTP "POST" request to the device authorization endpoint. The client constructs the request with the following parameters, encoded with the "application/x-www-form-urlencoded" content type:

client_id
REQUIRED. The client identifier as described in Section 2.2 of [RFC6749].

scope
OPTIONAL. The scope of the access request as described by Section 3.3 of [RFC6749].

For example, the client makes the following HTTPS request (line breaks are for display purposes only):

```
POST /device_authorization HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=459691054427
```

Parameters sent without a value MUST be treated as if they were omitted from the request. The authorization server MUST ignore unrecognized request parameters. Request and response parameters MUST NOT be included more than once.

3.2. Device Authorization Response

In response, the authorization server generates a device verification code and an end-user code that are valid for a limited time and includes them in the HTTP response body using the "application/json" format with a 200 (OK) status code. The response contains the following parameters:

device_code
: REQUIRED. The device verification code.

user_code
: REQUIRED. The end-user verification code.

verification_uri
: REQUIRED. The end-user verification URI on the authorization server. The URI should be short and easy to remember as end-users will be asked to manually type it into their user-agent.

verification_uri_complete
: OPTIONAL. A verification URI that includes the "user_code" (or other information with the same function as the "user_code"), designed for non-textual transmission.

expires_in
: OPTIONAL. The lifetime in seconds of the "device_code" and "user_code".

interval
: OPTIONAL. The minimum amount of time in seconds that the client SHOULD wait between polling requests to the token endpoint.

For example:


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

{
  "device_code":"GMMhmHCXhWEzkobqIHGG_EnNYYsAkukHspeYUk9E8",
  "user_code":"WDJB-MJHT",
  "verification_uri":"https://www.example.com/device",
```

```
      "verification_uri_complete":
           "https://www.example.com/device?user_code=WDJB-MJHT",
      "expires_in" : 1800,
      "interval": 5
    }
```

3.3. User Interaction

After receiving a successful Authorization Response, the client displays or otherwise communicates the "user_code" and the "verification_uri" to the end-user and instructs them to visit the URI in a user agent on a secondary device (for example, in a browser on their mobile phone), and enter the user code.

Figure 2: Example User Instruction

The authorizing user navigates to the "verification_uri" and authenticates with the authorization server in a secure TLS-protected session. The authorization server prompts the end-user to identify the device authorization session by entering the "user_code" provided by the client. The authorization server should then inform the user about the action they are undertaking and ask them to approve or deny the request. Once the user interaction is complete, the server informs the user to return to their device.

During the user interaction, the device continuously polls the token endpoint with the "device_code", as detailed in Section 3.4, until the user completes the interaction, the code expires, or another error occurs. The "device_code" is not intended for the end-user and MUST NOT be displayed or communicated.

Authorization servers supporting this specification MUST implement a

user interaction sequence that starts with the user navigating to "verification_uri" and continues with them supplying the "user_code" at some stage during the interaction. Other than that, the exact sequence and implementation of the user interaction is up to the authorization server and is out of scope of this specification.

It is NOT RECOMMENDED for authorization servers to include the user code in the verification URI ("verification_uri"), as this increases the length and complexity of the URI that the user must type. The next section documents user interaction with "verification_uri_complete", which is designed to carry this information.

3.3.1. Non-textual Verification URI Optimization

When "verification_uri_complete" is included in the Authorization Response (Section 3.2), clients MAY present this URI in a non-textual manner using any method that results in the browser being opened with the URI, such as with QR codes or NFC, to save the user typing the URI.

For usability reasons, it is RECOMMENDED for clients to still display the textual verification URI ("verification_uri") for users not able to use such a shortcut. Clients MUST still display the "user_code", as the authorization server may still require the user to confirm it to disambiguate devices, or as a remote phishing mitigation (See Section 5.3).

Figure 3: Example User Instruction with QR Code Representation of the Complete Verification URI

3.4. Device Access Token Request

After displaying instructions to the user, the client makes an Access Token Request to the token endpoint with a "grant_type" of "urn:ietf:params:oauth:grant-type:device_code". This is an extension grant type (as defined by Section 4.5 of [RFC6749]) with the following parameters:

grant_type
: REQUIRED. Value MUST be set to "urn:ietf:params:oauth:grant-type:device_code".

device_code
: REQUIRED. The device verification code, "device_code" from the Device Authorization Response, defined in Section 3.2.

client_id
: REQUIRED, if the client is not authenticating with the authorization server as described in Section 3.2.1. of [RFC6749].

For example, the client makes the following HTTPS request (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Adevice_code
&device_code=GMMhmHCXhWEzkobqIHGG_EnNYYsAkukHspeYUk9E8
&client_id=459691054427
```

If the client was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1 of [RFC6749]. Note that there are security implications of statically distributed client credentials, see Section 5.5.

The response to this request is defined in Section 3.5. Unlike other OAuth grant types, it is expected for the client to try the Access Token Request repeatedly in a polling fashion, based on the error code in the response.

3.5. Device Access Token Response

If the user has approved the grant, the token endpoint responds with a success response defined in Section 5.1 of [RFC6749]; otherwise it responds with an error, as defined in Section 5.2 of [RFC6749].

In addition to the error codes defined in Section 5.2 of [RFC6749], the following error codes are specified by the device flow for use in token endpoint responses:

authorization_pending
: The authorization request is still pending as the end-user hasn't yet completed the user interaction steps (Section 3.3). The client should repeat the Access Token Request to the token endpoint.

access_denied
: The end-user denied the authorization request.

slow_down
: The client is polling too quickly and should back off at a reasonable rate.

expired_token
: The "device_code" has expired. The client will need to make a new Device Authorization Request.

The error codes "authorization_pending" and "slow_down" are considered soft errors. The client should continue to poll the token endpoint by repeating the Device Token Request (Section 3.4) when receiving soft errors, increasing the time between polls if a "slow_down" error is received. Other error codes are considered hard errors; the client should stop polling and react accordingly, for example, by displaying an error to the user.

If the verification codes have expired, the server SHOULD respond with the error code "expired_token". Clients MAY then choose to start a new device authorization session.

The interval at which the client polls MUST NOT be more frequent than the "interval" parameter returned in the Device Authorization Response (see Section 3.2). If no interval was provided, the client MUST use a reasonable default polling interval.

The assumption of this specification is that the secondary device the user is authorizing the request on does not have a way to communicate back to the OAuth client. Only a one-way channel is required to make this flow useful in many scenarios. For example, an HTML application

on a TV that can only make outbound requests. If a return channel were to exist for the chosen user interaction interface, then the device MAY wait until notified on that channel that the user has completed the action before initiating the token request. Such behavior is, however, outside the scope of this specification.

4. Discovery Metadata

Support for the device flow MAY be declared in the OAuth 2.0 Authorization Server Metadata [I-D.ietf-oauth-discovery] with the following metadata:

device_authorization_endpoint
: OPTIONAL. URL of the authorization server's device authorization endpoint defined in Section 3.1.

5. Security Considerations

5.1. User Code Brute Forcing

Since the user code is typed by the user, shorter codes are more desirable for usability reasons. This means the entropy is typically less than would be used for the device code or other OAuth bearer token types where the code length does not impact usability. It is therefore recommended that the server rate-limit user code attempts. The user code SHOULD have enough entropy that when combined with rate limiting and other mitigations makes a brute-force attack infeasible.

A successful brute forcing of the user code would enable the attacker to authenticate with their own credentials and make an authorization grant to the device. This is the opposite scenario to an OAuth bearer token being brute forced, whereby the attacker gains control

of the victim's authorization grant. In some applications this attack may not make much economic sense, for example for a video app, the owner of the device may then be able to purchase movies with the attacker's account, however there are still privacy considerations in that case as well as other uses of the device flow whereby the granting account may be able to perform sensitive actions such as controlling the victim's device.

The precise length of the user code and the entropy contained within is at the discretion of the authorization server, which needs to consider the sensitivity of their specific protected resources, the practicality of the code length from a usability standpoint, and any mitigations that are in place such as rate-limiting, when determining the user code format.

5.2. Device Trustworthiness

Unlike other native application OAuth 2.0 flows, the device requesting the authorization is not the same as the device that the user grants access from. Thus, signals from the approving user's session and device are not relevant to the trustworthiness of the client device.

Note that if an authorization server used with this flow is malicious, then it could man-in-the middle the backchannel flow to another authorization server. In this scenario, the man-in-the-middle is not completely hidden from sight, as the end-user would end up on the authorization page of the wrong service, giving them an opportunity to notice that the authorization being requested is wrong. For this to be possible, the device manufacturer must either directly be the attacker, shipping a device intended to perform the man-in-the-middle attack, or be using an authorization server that is controlled by an attacker, possibly because the attacker compromised the authorization server used by the device. In part, the person purchasing the device is counting on it and its business partners to be trustworthy.

5.3. Remote Phishing

It is possible for the device flow to be initiated on a device in an

attacker's possession. For example, the attacker might send an email instructing the target user to visit the verification URL and enter the user code. To mitigate such an attack, it is RECOMMENDED to inform the user that they are authorizing a device during the user interaction step (see Section 3.3), and to confirm that the device is in their possession. The authorization server SHOULD display information about the device so that the person can notice if a software client was attempting to impersonating a hardware device.

For authorization servers that support the option specified in Section 3.3.1 for the client to append the user code to the authorization URI, it is particularly important to confirm that the device is in the user's possession, as the user no longer has to type the code manually. One possibility is to display the code during the authorization flow and asking the user to verify that the same code is being displayed on the device they are setting up.

The user code needs to have a long enough lifetime to be useable (allowing the user to retrieve their secondary device, navigate to the verification URI, login, etc.), but should be sufficiently short to limit the usability of a code obtained for phishing. This doesn't prevent a phisher presenting a fresh token, particularly in the case

they are interacting with the user in real time, but it does limit the viability of codes sent over email or SMS.

5.4. Session Spying

While the device is pending authorization, it may be possible for a malicious user to spy on the device user interface and hijack the session by completing the authorization faster than the user that initiated it. Devices SHOULD take into account the operating environment when considering how to communicate the code to the user to reduce the chances it will be observed by a malicious user.

5.5. Non-confidential Clients

Most device clients are incapable of being confidential clients, as secrets that are statically included as part of an app distributed to multiple users cannot be considered confidential. For such clients, the recommendations of Section 5.3.1 of [RFC6819] and Section 8.5 of

[RFC8252] apply.

5.6. Non-Visual Code Transmission

There is no requirement that the user code be displayed by the device visually. Other methods of one-way communication can potentially be used, such as text-to-speech audio, or Bluetooth Low Energy. To mitigate an attack in which a malicious user can bootstrap their credentials on a device not in their control, it is RECOMMENDED that any chosen communication channel only be accessible by people in close proximity. E.g., users who can see, or hear the device, or within range of a short-range wireless signal.

6. Usability Considerations

This section is a non-normative discussion of usability considerations.

6.1. User Code Recommendations

For many users, their nearest Internet-connected device will be their mobile phone, and typically these devices offer input methods that are more time consuming than a computer keyboard to change the case or input numbers. To improve usability (improving entry speed, and reducing retries), these limitations should be taken into account when selecting the user-code character set.

One way to improve input speed is to restrict the character set to case-insensitive A-Z characters, with no digits. These characters can typically be entered on a mobile keyboard without using modifier

keys. Further removing vowels to avoid randomly creating words results in the base-20 character set: "BCDFGHJKLMNPQRSTVWXZ". Dashes or other punctuation may be included for readability.

An example user code following this guideline, with an entropy of 20^8: "WDJB-MJHT".

Pure numeric codes are also a good choice for usability, especially for clients targeting locales where A-Z character keyboards are not used, though their length needs to be longer to maintain a high entropy.

An example numeric user code, with an entropy of 10^9: "019-450-730".

The server should ignore any characters like punctuation that are not in the user-code character set. Provided that the character set doesn't include characters of different case, the comparison should be case insensitive.

6.2. Non-Browser User Interaction

Devices and authorization servers MAY negotiate an alternative code transmission and user interaction method in addition to the one described in Section 3.3. Such an alternative user interaction flow could obviate the need for a browser and manual input of the code, for example, by using Bluetooth to transmit the code to the authorization server's companion app. Such interaction methods can utilize this protocol, as ultimately, the user just needs to identify the authorization session to the authorization server; however, user interaction other than via the verification URI is outside the scope of this specification.

7. IANA Considerations

7.1. OAuth URI Registration

This specification registers the following values in the IANA "OAuth URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

7.1.1. Registry Contents

- URN: urn:ietf:params:oauth:grant-type:device_code
- Common Name: Device flow grant type for OAuth 2.0
- Change controller: IESG
- Specification Document: Section 3.1 of [[this specification]]

7.2. OAuth Extensions Error Registration

This specification registers the following values in the IANA "OAuth Extensions Error Registry" registry [IANA.OAuth.Parameters]

established by [RFC6749].

7.2.1. Registry Contents

- Error name: authorization_pending
- Error usage location: Token endpoint response
- Related protocol extension: [[this specification]]
- Change controller: IETF
- Specification Document: Section 3.5 of [[this specification]]

- Error name: access_denied
- Error usage location: Token endpoint response
- Related protocol extension: [[this specification]]
- Change controller: IETF
- Specification Document: Section 3.5 of [[this specification]]

- Error name: slow_down
- Error usage location: Token endpoint response
- Related protocol extension: [[this specification]]
- Change controller: IETF
- Specification Document: Section 3.5 of [[this specification]]

- Error name: expired_token
- Error usage location: Token endpoint response
- Related protocol extension: [[this specification]]
- Change controller: IETF
- Specification Document: Section 3.5 of [[this specification]]

7.3. OAuth 2.0 Authorization Server Metadata

This specification registers the following values in the IANA "OAuth 2.0 Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [I-D.ietf-oauth-discovery].

7.3.1. Registry Contents

- Metadata name: device_authorization_endpoint
- Metadata Description: The Device Authorization Endpoint.
- Change controller: IESG
- Specification Document: Section 4 of [[this specification]]

8. Normative References

[I-D.ietf-oauth-discovery]
Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", draft-ietf-oauth-discovery-10 (work in progress), March 2018.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

Appendix A. Acknowledgements

The starting point for this document was the Internet-Draft draft-recordon-oauth-v2-device, authored by David Recordon and Brent Goldman, which itself was based on content in draft versions of the OAuth 2.0 protocol specification removed prior to publication due to a then lack of sufficient deployment expertise. Thank you to the OAuth working group members who contributed to those earlier drafts.

This document was produced in the OAuth working group under the chairpersonship of Rifaat Shekh-Yusef and Hannes Tschofenig with Benjamin Kaduk, Kathleen Moriarty, and Eric Rescorla serving as Security Area Directors.

The following individuals contributed ideas, feedback, and wording that shaped and formed the final specification:

Brian Campbell, Roshni Chandrashekhar, Eric Fazendin, Torsten Lodderstedt, James Manger, Breno de Medeiros, Simon Moffatt, Stein Myrseth, Justin Richer, Nat Sakimura, Andrew Sciberras, Marius Scurtescu, Ken Wang, and Steven E. Wright.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-10

- Added a missing definition of access_denied for use on the token endpoint.
- Corrected text documenting which error code should be returned for expired tokens (it's "expired_token", not "invalid_grant").
- Corrected section reference to RFC 8252 (the section numbers had changed after the initial reference was made).
- Fixed line length of one diagram (was causing xml2rfc warnings).
- Added line breaks so the URN grant_type is presented on an unbroken line.
- Typos fixed and other stylistic improvements.

-09

- Addressed review comments by Security Area Director Eric Rescorla about the potential of a confused deputy attack.

-08

- Expanded the User Code Brute Forcing section to include more detail on this attack.

-07

- Replaced the "user_code" URI parameter optimization with verification_uri_complete following the IETF99 working group discussion.

- Added security consideration about spying.
- Required that device_code not be shown.
- Added text regarding a minimum polling interval.

-06

- Clarified usage of the "user_code" URI parameter optimization following the IETF98 working group discussion.

-05

- response_type parameter removed from authorization request.
- Added option for clients to include the user_code on the verification URI.
- Clarified token expiry, and other nits.

-04

- Security & Usability sections. OAuth Discovery Metadata.

-03

- device_code is now a URN. Added IANA Considerations

-02

- Added token request & response specification.

-01

- Applied spelling and grammar corrections and added the Document History appendix.

-00

- Initial working group draft based on draft-recordon-oauth-v2-device.

Authors' Addresses

William Denniss
Google

1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Email: wdenniss@google.com
URI: http://wdenniss.com/device-flow

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at