Internet Engineering Task Force (IETF) W. Denniss

Request for Comments: 8628 Google

Category: Standards Track J. Bradley

ISSN: 2070-1721 Ping Identity

M. Jones

Microsoft

H. Tschofenig

ARM Limited

August 2019

OAuth 2.0 Device Authorization Grant

Abstract

The OAuth 2.0 device authorization grant is designed for Internet-

connected devices that either lack a browser to perform a user-agent-

based authorization or are input constrained to the extent that

requiring the user to input text in order to authenticate during the

authorization flow is impractical. It enables OAuth clients on such

devices (like smart TVs, media consoles, digital picture frames, and

printers) to obtain user authorization to access protected resources

by using a user agent on a separate device.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force

(IETF). It represents the consensus of the IETF community. It has

received public review and has been approved for publication by the

Internet Engineering Steering Group (IESG). Further information on

Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata,

and how to provide feedback on it may be obtained at

https://www.rfc-editor.org/info/rfc8628.

Denniss, et al. Standards Track [Page 1]

RFC 8628 OAuth 2.0 Device Grant August 2019

Copyright Notice

Copyright (c) 2019 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

(https://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. Terminology . . . . . . . . . . . . . . . . . . . . . . . . . 5

3. Protocol . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1. Device Authorization Request . . . . . . . . . . . . . . 5

3.2. Device Authorization Response . . . . . . . . . . . . . . 7

3.3. User Interaction . . . . . . . . . . . . . . . . . . . . 8

3.3.1. Non-Textual Verification URI Optimization . . . . . . 9

3.4. Device Access Token Request . . . . . . . . . . . . . . . 10

3.5. Device Access Token Response . . . . . . . . . . . . . . 11

4. Discovery Metadata . . . . . . . . . . . . . . . . . . . . . 12

5. Security Considerations . . . . . . . . . . . . . . . . . . . 12

5.1. User Code Brute Forcing . . . . . . . . . . . . . . . . . 12

5.2. Device Code Brute Forcing . . . . . . . . . . . . . . . . 13

5.3. Device Trustworthiness . . . . . . . . . . . . . . . . . 13

5.4. Remote Phishing . . . . . . . . . . . . . . . . . . . . . 14

5.5. Session Spying . . . . . . . . . . . . . . . . . . . . . 15

5.6. Non-Confidential Clients . . . . . . . . . . . . . . . . 15

5.7. Non-Visual Code Transmission . . . . . . . . . . . . . . 15

6. Usability Considerations . . . . . . . . . . . . . . . . . . 16

6.1. User Code Recommendations . . . . . . . . . . . . . . . . 16

6.2. Non-Browser User Interaction . . . . . . . . . . . . . . 17

7. IANA Considerations . . . . . . . . . . . . . . . . . . . . . 17

7.1. OAuth Parameter Registration . . . . . . . . . . . . . . 17

7.2. OAuth URI Registration . . . . . . . . . . . . . . . . . 17

7.3. OAuth Extensions Error Registration . . . . . . . . . . . 18

7.4. OAuth Authorization Server Metadata . . . . . . . . . . . 18

8. Normative References . . . . . . . . . . . . . . . . . . . . 19

Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . 20

Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . 21

Denniss, et al. Standards Track [Page 2]

RFC 8628 OAuth 2.0 Device Grant August 2019

1. Introduction

This OAuth 2.0 [RFC6749] protocol extension enables OAuth clients to

request user authorization from applications on devices that have

limited input capabilities or lack a suitable browser. Such devices

include smart TVs, media consoles, picture frames, and printers,

which lack an easy input method or a suitable browser required for

traditional OAuth interactions. The authorization flow defined by

this specification, sometimes referred to as the "device flow",

instructs the user to review the authorization request on a secondary

device, such as a smartphone, which does have the requisite input and

browser capabilities to complete the user interaction.

The device authorization grant 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" [RFC8252].

The operating requirements for using this authorization grant type

are:

(1) The device is already connected to the Internet.

(2) The device is able to make outbound HTTPS requests.

(3) The device is able to display or otherwise communicate a URI and

code sequence to the user.

(4) The user has a secondary device (e.g., personal computer or

smartphone) from which they can process the request.

As the device authorization grant does not require two-way

communication between the OAuth client on the device and the user

agent (unlike other OAuth 2 grant types, such as the authorization

code and implicit grant types), it supports several use cases that

cannot be served by those other approaches.

Instead of interacting directly with the end user's user agent (i.e.,

browser), the device client instructs the end user to use another

computer or device and connect to the authorization server to approve

the access request. Since the protocol supports clients that can't

receive incoming requests, clients poll the authorization server

repeatedly until the end user completes the approval process.

Denniss, et al. Standards Track [Page 3]

RFC 8628 OAuth 2.0 Device Grant August 2019

The device client typically chooses the set of authorization servers

to support (i.e., its own authorization server or those of providers

with which it has relationships). It is common for the device client

to support only one authorization server, such as in the case of a TV

application for a specific media provider that supports only that

media provider's authorization server. The user may not yet have an

established relationship with that authorization provider, though one

can potentially be set up during the authorization flow.

+----------+ +----------------+

| |>---(A)-- Client Identifier --->| |

| | | |

| |<---(B)-- Device Code, ---<| |

| | User Code, | |

| Device | & Verification URI | |

| Client | | |

| | [polling] | |

| |>---(E)-- Device Code --->| |

| | & Client Identifier | |

| | | Authorization |

| |<---(F)-- Access Token ---<| Server |

+----------+ (& Optional Refresh Token) | |

v | |

: | |

(C) User Code & Verification URI | |

: | |

v | |

+----------+ | |

| End User | | |

| at |<---(D)-- End user reviews --->| |

| Browser | authorization request | |

+----------+ +----------------+

Figure 1: Device Authorization Flow

The device authorization 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 device code and an end-user

code and provides the end-user verification URI.

(C) The client instructs the end user to use a user agent on another

device and visit the provided end-user verification URI. The

client provides the user with the end-user code to enter in

order to review the authorization request.

Denniss, et al. Standards Track [Page 4]

RFC 8628 OAuth 2.0 Device Grant August 2019

(D) The authorization server authenticates the end user (via the

user agent), and prompts the user to input the user code

provided by the device client. The authorization server

validates the user code provided by the user, and prompts the

user to accept or decline the request.

(E) While the end user reviews the client's request (step D), the

client repeatedly polls the authorization server to find out if

the user completed the user authorization step. The client

includes the device code and its client identifier.

(F) The authorization server validates the device code provided by

the client and responds with the access token if the client is

granted access, an error if they are denied access, or an

indication that the client should continue to poll.

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

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all

capitals, as shown here.

3. Protocol

3.1. Device Authorization Request

This specification defines a new OAuth endpoint: the device

authorization endpoint. This is separate from the OAuth

authorization endpoint defined in [RFC6749] with which the user

interacts via a user agent (i.e., a browser). By comparison, when

using the device authorization endpoint, the OAuth client on the

device interacts with the authorization server directly without

presenting the request in a user agent, and the end user authorizes

the request on a separate device. This interaction is defined as

follows.

The client initiates the authorization flow by requesting a set of

verification codes from the authorization server by making an HTTP

"POST" request to the device authorization endpoint.

Denniss, et al. Standards Track [Page 5]

RFC 8628 OAuth 2.0 Device Grant August 2019

The client makes a device authorization request to the device

authorization endpoint by including the following parameters using

the "application/x-www-form-urlencoded" format, per Appendix B of

[RFC6749], with a character encoding of UTF-8 in the HTTP request

entity-body:

client\_id

REQUIRED if the client is not authenticating with the

authorization server as described in Section 3.2.1. of [RFC6749].

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

scope

OPTIONAL. The scope of the access request as defined by

Section 3.3 of [RFC6749].

For example, the client makes the following HTTPS request:

POST /device\_authorization HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

client\_id=1406020730&scope=example\_scope

All requests from the device MUST use the Transport Layer Security

(TLS) protocol [RFC8446] and implement the best practices of BCP 195

[RFC7525].

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.

The client authentication requirements of Section 3.2.1 of [RFC6749]

apply to requests on this endpoint, which means that confidential

clients (those that have established client credentials) authenticate

in the same manner as when making requests to the token endpoint, and

public clients provide the "client\_id" parameter to identify

themselves.

Due to the polling nature of this protocol (as specified in

Section 3.4), care is needed to avoid overloading the capacity of the

token endpoint. To avoid unneeded requests on the token endpoint,

the client SHOULD only commence a device authorization request when

prompted by the user and not automatically, such as when the app

starts or when the previous authorization session expires or fails.

Denniss, et al. Standards Track [Page 6]

RFC 8628 OAuth 2.0 Device Grant August 2019

3.2. Device Authorization Response

In response, the authorization server generates a unique 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 [RFC8259] 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"),

which is designed for non-textual transmission.

expires\_in

REQUIRED. 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. If no

value is provided, clients MUST use 5 as the default.

For example:

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

{

"device\_code": "GmRhmhcxhwAzkoEqiMEg\_DnyEysNkuNhszIySk9eS",

"user\_code": "WDJB-MJHT",

"verification\_uri": "https://example.com/device",

"verification\_uri\_complete":

"https://example.com/device?user\_code=WDJB-MJHT",

"expires\_in": 1800,

"interval": 5

}

Denniss, et al. Standards Track [Page 7]

RFC 8628 OAuth 2.0 Device Grant August 2019

In the event of an error (such as an invalidly configured client),

the authorization server responds in the same way as the token

endpoint specified in Section 5.2 of [RFC6749].

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.

+-----------------------------------------------+

| |

| Using a browser on another device, visit: |

| https://example.com/device |

| |

| And enter the code: |

| WDJB-MJHT |

| |

+-----------------------------------------------+

Figure 2: Example User Instruction

The authorizing user navigates to the "verification\_uri" and

authenticates with the authorization server in a secure TLS-protected

[RFC8446] 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 instructs 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

directly; thus, it should not be displayed during the interaction to

avoid confusing the end user.

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; for example, the authorization server may

enable new users to sign up for an account during the authorization

flow or add additional security verification steps.

Denniss, et al. Standards Track [Page 8]

RFC 8628 OAuth 2.0 Device Grant August 2019

It is NOT RECOMMENDED for authorization servers to include the user

code ("user\_code") in the verification URI ("verification\_uri"), as

this increases the length and complexity of the URI that the user

must type. While the user must still type a similar number of

characters with the "user\_code" separated, once they successfully

navigate to the "verification\_uri", any errors in entering the code

can be highlighted by the authorization server to improve the user

experience. The next section documents the user interaction with

"verification\_uri\_complete", which is designed to carry both pieces

of 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 (Quick Response) codes or NFC (Near Field

Communication), to save the user from typing the URI.

For usability reasons, it is RECOMMENDED for clients to still display

the textual verification URI ("verification\_uri") for users who are

not able to use such a shortcut. Clients MUST still display the

"user\_code", as the authorization server will require the user to

confirm it to disambiguate devices or as remote phishing mitigation

(see Section 5.4).

If the user starts the user interaction by navigating to

"verification\_uri\_complete", then the user interaction described in

Section 3.3 is still followed, with the optimization that the user

does not need to type in the "user\_code". The server SHOULD display

the "user\_code" to the user and ask them to verify that it matches

the "user\_code" being displayed on the device to confirm they are

authorizing the correct device. As before, in addition to taking

steps to confirm the identity of the device, the user should also be

afforded the choice to approve or deny the authorization request.

Denniss, et al. Standards Track [Page 9]

RFC 8628 OAuth 2.0 Device Grant August 2019

+-------------------------------------------------+

| |

| Scan the QR code or, using +------------+ |

| a browser on another device, |[\_].. . [\_]| |

| visit: | . .. . .| |

| https://example.com/device | . . . ....| |

| |. . . . | |

| And enter the code: |[\_]. ... . | |

| WDJB-MJHT +------------+ |

| |

+-------------------------------------------------+

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 creates an

access token request and sends it to the token endpoint (as defined

by Section 3.2 of [RFC6749]) 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]) created by this

specification, 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].

The client identifier as described in Section 2.2 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=GmRhmhcxhwAzkoEqiMEg\_DnyEysNkuNhszIySk9eS

&client\_id=1406020730

Denniss, et al. Standards Track [Page 10]

RFC 8628 OAuth 2.0 Device Grant August 2019

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 5.6.

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 for use with the device

authorization grant 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 (a process known as polling). Before each new request,

the client MUST wait at least the number of seconds specified by

the "interval" parameter of the device authorization response (see

Section 3.2), or 5 seconds if none was provided, and respect any

increase in the polling interval required by the "slow\_down"

error.

slow\_down

A variant of "authorization\_pending", the authorization request is

still pending and polling should continue, but the interval MUST

be increased by 5 seconds for this and all subsequent requests.

access\_denied

The authorization request was denied.

expired\_token

The "device\_code" has expired, and the device authorization

session has concluded. The client MAY commence a new device

authorization request but SHOULD wait for user interaction before

restarting to avoid unnecessary polling.

Denniss, et al. Standards Track [Page 11]

RFC 8628 OAuth 2.0 Device Grant August 2019

The "authorization\_pending" and "slow\_down" error codes define

particularly unique behavior, as they indicate that the OAuth client

should continue to poll the token endpoint by repeating the token

request (implementing the precise behavior defined above). If the

client receives an error response with any other error code, it MUST

stop polling and SHOULD react accordingly, for example, by displaying

an error to the user.

On encountering a connection timeout, clients MUST unilaterally

reduce their polling frequency before retrying. The use of an

exponential backoff algorithm to achieve this, such as doubling the

polling interval on each such connection timeout, is RECOMMENDED.

The assumption of this specification is that the separate device on

which the user is authorizing the request does not have a way to

communicate back to the device with the OAuth client. This protocol

only requires a one-way channel in order to maximize the viability of

the protocol in restricted environments, like an application running

on a TV that is only capable of 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 (as an

alternative to polling). Such behavior is, however, outside the

scope of this specification.

4. Discovery Metadata

Support for this protocol is declared in OAuth 2.0 Authorization

Server Metadata [RFC8414] as follows. The value

"urn:ietf:params:oauth:grant-type:device\_code" is included in values

of the "grant\_types\_supported" key, and the following new key value

pair is added:

device\_authorization\_endpoint

OPTIONAL. URL of the authorization server's device authorization

endpoint, as 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.

Therefore, it is recommended that the server rate-limit user code

attempts.

Denniss, et al. Standards Track [Page 12]

RFC 8628 OAuth 2.0 Device Grant August 2019

The user code SHOULD have enough entropy that, when combined with

rate-limiting and other mitigations, a brute-force attack becomes

infeasible. For example, it's generally held that 128-bit symmetric

keys for encryption are seen as good enough today because an attacker

has to put in 2^96 work to have a 2^-32 chance of guessing correctly

via brute force. The rate-limiting and finite lifetime on the user

code place an artificial limit on the amount of work an attacker can

"do". If, for instance, one uses an 8-character base 20 user code

(with roughly 34.5 bits of entropy), the rate-limiting interval and

validity period would need to only allow 5 attempts in order to get

the same 2^-32 probability of success by random guessing.

A successful brute forcing of the user code would enable the attacker

to approve the authorization grant with their own credentials, after

which the device would receive a device authorization grant linked to

the attacker's account. This is the opposite scenario to an OAuth

bearer token being brute forced, whereby the attacker gains control

of the victim's authorization grant. Such attacks may not always

make economic sense. For example, for a video app, the device owner

may then be able to purchase movies using the attacker's account

(though even in this case a privacy risk would still remain and thus

is important to protect against). Furthermore, some uses of the

device flow give the granting account the ability to perform actions

that need to be protected, such as controlling the 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 Code Brute Forcing

An attacker who guesses the device code would be able to potentially

obtain the authorization code once the user completes the flow. As

the device code is not displayed to the user and thus there are no

usability considerations on the length, a very high entropy code

SHOULD be used.

5.3. Device Trustworthiness

Unlike other native application OAuth 2.0 flows, the device

requesting the authorization is not the same as the device from which

the user grants access. Thus, signals from the approving user's

session and device are not always relevant to the trustworthiness of

the client device.

Denniss, et al. Standards Track [Page 13]

RFC 8628 OAuth 2.0 Device Grant August 2019

Note that if an authorization server used with this flow is

malicious, then it could perform a man-in-the-middle attack on 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 URL in the browser's

address bar is wrong. For this to be possible, the device

manufacturer must either be the attacker and 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 the

manufacturer and its business partners to be trustworthy.

5.4. Remote Phishing

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

attacker's possession. For example, an 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 user could notice if a

software client was attempting to impersonate a hardware device.

For authorization servers that support the

"verification\_uri\_complete" optimization discussed in Section 3.3.1,

it is particularly important to confirm that the device is in the

user's possession, as the user no longer has to type in the code

being displayed on the device manually. One suggestion is to display

the code during the authorization flow and ask the user to verify

that the same code is currently 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, log in, etc.) but should be sufficiently short

to limit the usability of a code obtained for phishing. This doesn't

prevent a phisher from presenting a fresh token, particularly if they

are interacting with the user in real time, but it does limit the

viability of codes sent over email or text message.

Denniss, et al. Standards Track [Page 14]

RFC 8628 OAuth 2.0 Device Grant August 2019

5.5. Session Spying

While the device is pending authorization, it may be possible for a

malicious user to physically spy on the device user interface (by

viewing the screen on which it's displayed, for example) 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.

5.6. Non-Confidential Clients

Device clients are generally incapable of maintaining the

confidentiality of their credentials, as users in possession of the

device can reverse-engineer it and extract the credentials.

Therefore, unless additional measures are taken, they should be

treated as public clients (as defined by Section 2.1 of [RFC6749]),

which are susceptible to impersonation. The security considerations

of Section 5.3.1 of [RFC6819] and Sections 8.5 and 8.6 of [RFC8252]

apply to such clients.

The user may also be able to obtain the "device\_code" and/or other

OAuth bearer tokens issued to their client, which would allow them to

use their own authorization grant directly by impersonating the

client. Given that the user in possession of the client credentials

can already impersonate the client and create a new authorization

grant (with a new "device\_code"), this doesn't represent a separate

impersonation vector.

5.7. 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, for example, users who can see or hear the device.

Denniss, et al. Standards Track [Page 15]

RFC 8628 OAuth 2.0 Device Grant August 2019

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; 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), the limitations of such devices 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, "WDJB-MJHT", contains

8 significant characters and has dashes added for end-user

readability. The resulting entropy is 20^8.

Pure numeric codes are also a good choice for usability, especially

for clients targeting locales where A-Z character keyboards are not

used, though the length of such a code needs to be longer to maintain

high entropy.

An example numeric user code that contains 9 significant digits and

dashes added for end-user readability with an entropy of 10^9 is

"019-450-730".

When processing the inputted user code, the server should strip

dashes and other punctuation that it added for readability (making

the inclusion of such punctuation by the user optional). For codes

using only characters in the A-Z range, as with the base 20 charset

defined above, the user's input should be uppercased before a

comparison to account for the fact that the user may input the

equivalent lowercase characters. Further stripping of all characters

outside the chosen character set is recommended to reduce instances

where an errantly typed character (like a space character)

invalidates otherwise valid input.

Denniss, et al. Standards Track [Page 16]

RFC 8628 OAuth 2.0 Device Grant August 2019

It is RECOMMENDED to avoid character sets that contain two or more

characters that can easily be confused with each other, like "0" and

"O" or "1", "l" and "I". Furthermore, to the extent practical, when

a character set contains a character that may be confused with

characters outside the character set, a character outside the set MAY

be substituted with the one in the character set with which it is

commonly confused; for example, "O" may be substituted for "0" when

using the numerical 0-9 character set.

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 through the verification URI is outside the

scope of this specification.

7. IANA Considerations

7.1. OAuth Parameter Registration

This specification registers the following values in the IANA "OAuth

Parameters" registry [IANA.OAuth.Parameters] established by

[RFC6749].

Name: device\_code

Parameter Usage Location: token request

Change Controller: IESG

Reference: Section 3.4 of RFC 8628

7.2. OAuth URI Registration

This specification registers the following values in the IANA "OAuth

URI" registry [IANA.OAuth.Parameters] established by [RFC6755].

URN: urn:ietf:params:oauth:grant-type:device\_code

Common Name: Device Authorization Grant Type for OAuth 2.0

Change Controller: IESG

Specification Document: Section 3.4 of RFC 8628

Denniss, et al. Standards Track [Page 17]

RFC 8628 OAuth 2.0 Device Grant August 2019

7.3. OAuth Extensions Error Registration

This specification registers the following values in the IANA "OAuth

Extensions Error Registry" registry [IANA.OAuth.Parameters]

established by [RFC6749].

Name: authorization\_pending

Usage Location: Token endpoint response

Protocol Extension: RFC 8628

Change Controller: IETF

Reference: Section 3.5 of RFC 8628

Name: access\_denied

Usage Location: Token endpoint response

Protocol Extension: RFC 8628

Change Controller: IETF

Reference: Section 3.5 of RFC 8628

Name: slow\_down

Usage Location: Token endpoint response

Protocol Extension: RFC 8628

Change Controller: IETF

Reference: Section 3.5 of RFC 8628

Name: expired\_token

Usage Location: Token endpoint response

Protocol Extension: RFC 8628

Change Controller: IETF

Reference: Section 3.5 of RFC 8628

7.4. OAuth Authorization Server Metadata

This specification registers the following values in the IANA "OAuth

Authorization Server Metadata" registry [IANA.OAuth.Parameters]

established by [RFC8414].

Metadata name: device\_authorization\_endpoint

Metadata Description: URL of the authorization server's device

authorization endpoint

Change Controller: IESG

Reference: Section 4 of RFC 8628

Denniss, et al. Standards Track [Page 18]

RFC 8628 OAuth 2.0 Device Grant August 2019

8. Normative References

[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>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,

"Recommendations for Secure Use of Transport Layer

Security (TLS) and Datagram Transport Layer Security

(DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May

2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC

2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data

Interchange Format", STD 90, RFC 8259,

DOI 10.17487/RFC8259, December 2017,

<https://www.rfc-editor.org/info/rfc8259>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0

Authorization Server Metadata", RFC 8414,

DOI 10.17487/RFC8414, June 2018,

<https://www.rfc-editor.org/info/rfc8414>.

Denniss, et al. Standards Track [Page 19]

RFC 8628 OAuth 2.0 Device Grant August 2019

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol

Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,

<https://www.rfc-editor.org/info/rfc8446>.

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:

Ben Campbell, Brian Campbell, Roshni Chandrashekhar, Alissa Cooper,

Eric Fazendin, Benjamin Kaduk, Jamshid Khosravian, Mirja Kuehlewind,

Torsten Lodderstedt, James Manger, Dan McNulty, Breno de Medeiros,

Alexey Melnikov, Simon Moffatt, Stein Myrseth, Emond Papegaaij,

Justin Richer, Adam Roach, Nat Sakimura, Andrew Sciberras, Marius

Scurtescu, Filip Skokan, Robert Sparks, Ken Wang, Christopher Wood,

Steven E. Wright, and Qin Wu.

Denniss, et al. Standards Track [Page 20]

RFC 8628 OAuth 2.0 Device Grant August 2019

Authors' Addresses

William Denniss

Google

1600 Amphitheatre Pkwy

Mountain View, CA 94043

United States of America

Email: wdenniss@google.com

URI: https://wdenniss.com/deviceflow

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

Denniss, et al. Standards Track [Page 21]