Stream: Internet Engineering Task Force (IETF)
RFC: 9470
Category: Standards Track
Published: August 2023
ISSN: 2070-1721
Authors: V. Bertocci *Auth0/Okta*
B. Campbell *Ping Identity*

RFC 9470
OAuth 2.0 Step Up Authentication Challenge Protocol

Abstract

It is not uncommon for resource servers to require different authentication strengths or recentness according to the characteristics of a request. This document introduces a mechanism that resource servers can use to signal to a client that the authentication event associated with the access token of the current request does not meet its authentication requirements and, further, how to meet them. This document also codifies a mechanism for a client to request that an authorization server achieve a specific authentication strength or recentness when processing an authorization request.

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/rfc9470.

Copyright Notice

Table of Contents

1. Introduction

In simple API authorization scenarios, an authorization server will determine what authentication technique to use to handle a given request on the basis of aspects such as the scopes requested, the resource, the identity of the client, and other characteristics known at

provisioning time. Although that approach is viable in many situations, it falls short in several important circumstances. Consider, for instance, an eCommerce API requiring different authentication strengths depending on whether the item being purchased exceeds a certain threshold, dynamically estimated by the API itself using a logic that is opaque to the authorization server. An API might also determine that a more recent user authentication is required based on its own risk evaluation of the API request.

This document extends the collection of error codes defined by [RFC6750] with a new value, `insufficient_user_authentication`, which can be used by resource servers to signal to the client that the authentication event associated with the access token presented with the request does not meet the authentication requirements of the resource server. This document also introduces `acr_values` and `max_age` parameters for the `Bearer` authentication scheme challenge defined by [RFC6750]. The resource server can use these parameters to explicitly communicate to the client the required authentication strength or recentness.

The client can use that information to reach back to the authorization server with an authorization request that specifies the authentication requirements indicated by the protected resource. This is accomplished by including the `acr_values` or `max_age` authorization request parameters as defined in [OIDC].

Those extensions will make it possible to implement interoperable step up authentication with minimal work from resource servers, clients, and authorization servers.

1.1. Conventions and 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.

This specification uses the terms "access token", "authorization server", "authorization endpoint", "authorization request", "client", "protected resource", and "resource server" defined by "The OAuth 2.0 Authorization Framework" [RFC6749].

2. Protocol Overview

The following is an end-to-end sequence of a typical step up authentication scenario implemented according to this specification. The scenario assumes that, before the sequence described below takes place, the client already obtained an access token for the protected resource.

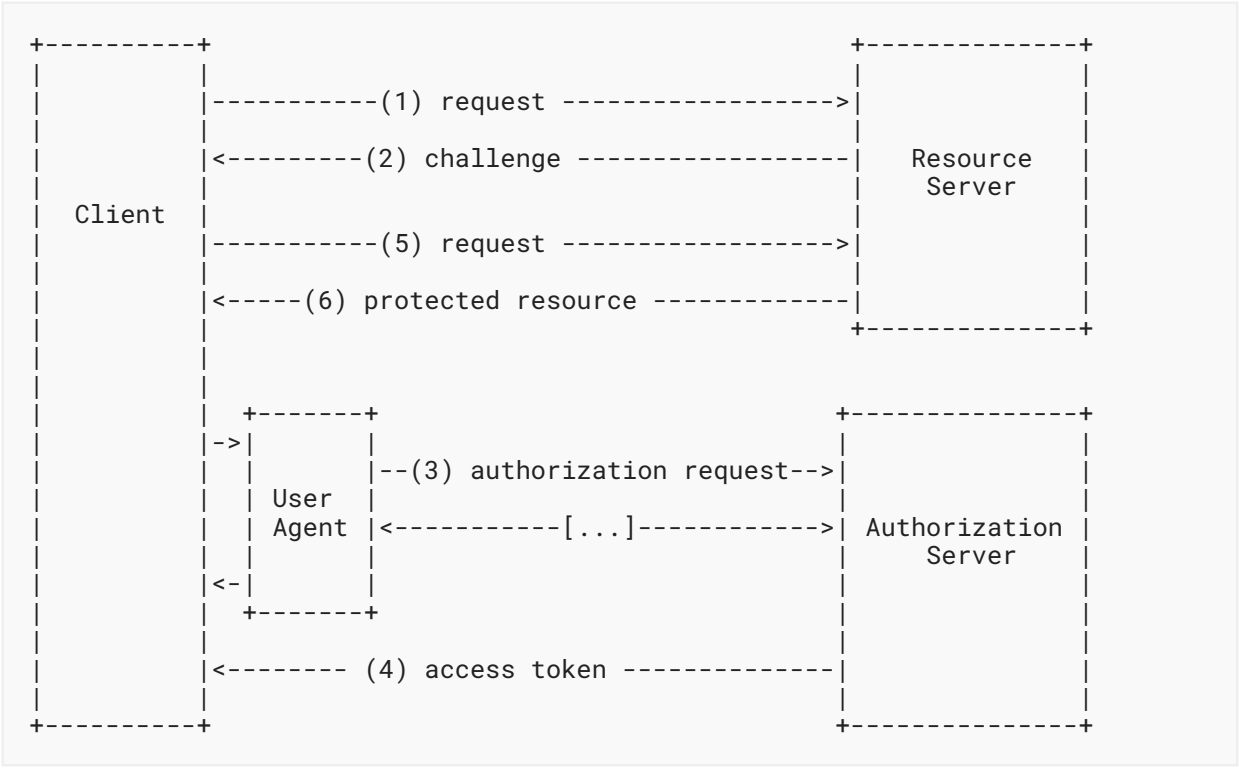

Figure 1: Abstract Protocol Flow

1. The client requests a protected resource, presenting an access token.
2. The resource server determines that the circumstances in which the presented access token was obtained offer insufficient authentication strength and/or recentness; hence, it denies the request and returns a challenge describing (using a combination of `acr_values` and `max_age`) what authentication requirements must be met for the resource server to authorize a request.
3. The client directs the user agent to the authorization server with an authorization request that includes the `acr_values` and/or `max_age` indicated by the resource server in the previous step.
4. Whatever sequence required by the grant of choice plays out; this will include the necessary steps to authenticate the user in accordance with the `acr_values` and/or `max_age` values of the authorization request. Then, the authorization server returns a new access token to the client. The new access token contains or references information about the authentication event.
5. The client repeats the request from step 1, presenting the newly obtained access token.
6. The resource server finds that the user authentication performed during the acquisition of the new access token complies with its requirements and returns the representation of the requested protected resource.

The validation operations mentioned in steps 2 and 6 imply that the resource server has a way of evaluating the authentication that occurred during the process by which the access token was obtained. In the context of this document, the assessment by the resource server of the specific authentication method used to obtain a token for the requested resource is called an "authentication level". This document will describe how the resource server can perform this assessment of an authentication level when the access token is a JSON Web Token (JWT) [RFC9068] or is validated via introspection [RFC7662]. Other methods of determining the authentication level by which the access token was obtained are possible, per agreement by the authorization server and the protected resource, but they are beyond the scope of this specification. Given an authentication level of a token, the resource server determines whether it meets the security criteria for the requested resource.

The terms "authentication level" and "step up" are metaphors in this specification. These metaphors do not suggest that there is an absolute hierarchy of authentication methods expressed in interoperable fashion. The notion of a level emerges from the fact that the resource server may only want to accept certain authentication methods. When presented with a token derived from a particular authentication method (i.e., a given authentication level) that it does not want to accept (i.e., below the threshold or level it will accept), the resource server seeks to step up (i.e., renegotiate) from the current authentication level to one that it may accept. The "step up" metaphor is intended to convey a shift from the original authentication level to one that is acceptable to the resource server.

Although the case in which the new access token supersedes old tokens by virtue of a higher authentication level is common, in line with the connotation of the term "step up authentication", it is important to keep in mind that this might not necessarily hold true in the general case. For example, for a particular request, a resource server might require a higher authentication level and a shorter validity, resulting in a token suitable for one-off calls but leading to frequent prompts: hence, offering a suboptimal user experience if the token is reused for routine operations. In such a scenario, the client would be better served by keeping both the old tokens, which are associated with a lower authentication level, and the new one: selecting the appropriate token for each API call. This is not a new requirement for clients, as incremental consent and least-privilege principles will require similar heuristics for managing access tokens associated with different scopes and permission levels. This document does not recommend any specific token-caching strategy: that choice will be dependent on the characteristics of every particular scenario and remains application-dependent as in the core OAuth cases. Also recall that OAuth 2.0 [RFC6749] assumes access tokens are treated as opaque by clients. The token format might be unreadable to the client or might change at any time to become unreadable. So, during the course of any token-caching strategy, a client must not attempt to inspect the content of the access token to determine the associated authentication information or other details (see Section 6 of [RFC9068] for a more detailed discussion).

3. Authentication Requirements Challenge

This specification introduces a new error code value for the challenge of the `Bearer` authentication scheme's `error` parameter (from [RFC6750]) and other OAuth authentication schemes, such as those seen in [RFC9449], which use the same `error` parameter:

`insufficient_user_authentication`: The authentication event associated with the access token presented with the request does not meet the authentication requirements of the protected resource.

Note: the logic through which the resource server determines that the current request does not meet the authentication requirements of the protected resource, and associated functionality (such as expressing, deploying and publishing such requirements), is out of scope for this document.

Furthermore, this specification defines the following `WWW-Authenticate` auth-param values for those OAuth authentication schemes to convey the authentication requirements back to the client.

`acr_values`: A space-separated string listing the authentication context class reference values in order of preference. The protected resource requires one of these values for the authentication event associated with the access token. As defined in Section 1.2 of [OIDC], the authentication context conveys information about how authentication takes place (e.g., what authentication method(s) or assurance level to meet).

`max_age`: This value indicates the allowable elapsed time in seconds since the last active authentication event associated with the access token. An active authentication event entails a user interacting with the authorization server in response to an authentication prompt. Note that, while the auth-param value can be conveyed as a token or quoted-string (see Section 11.2 of [RFC9110]), it has to represent a non-negative integer.

Figure 2 is an example of a `Bearer` authentication scheme challenge with the `WWW-Authenticate` header using:

- the `insufficient_user_authentication` error code value to inform the client that the access token presented is not sufficient to gain access to the protected resource, and
- the `acr_values` parameter to let the client know that the expected authentication level corresponds to the authentication context class reference identified by `myACR`.

Note that while this specification only defines usage of the above auth-params with the `insufficient_user_authentication` error code, it does not preclude future specifications or profiles from defining their usage with other error codes.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer error="insufficient_user_authentication",
  error_description="A different authentication level is required",
  acr_values="myACR"
```

Figure 2: Authentication Requirements Challenge Indicating `acr_values`

The example in Figure 3 shows a challenge informing the client that the last active authentication event associated with the presented access token is too old and a more recent authentication is needed.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer error="insufficient_user_authentication",
  error_description="More recent authentication is required",
  max_age="5"
```

Figure 3: Authentication Requirements Challenge Indicating `max_age`

The auth-params `max_age` and `acr_values` **MAY** both occur in the same challenge if the resource server needs to express requirements about both recency and authentication level. If the resource server determines that the request is also lacking the scopes required by the requested resource, it **MAY** include the `scope` attribute with the value necessary to access the protected resource, as described in Section 3.1 of [RFC6750].

4. Authorization Request

A client receiving a challenge from the resource server carrying the `insufficient_user_authentication` error code **SHOULD** parse the `WWW-Authenticate` header for `acr_values` and `max_age` and use them, if present, in constructing an authorization request. This request is then conveyed to the authorization server's authorization endpoint via the user agent in order to obtain a new access token complying with the corresponding requirements. The `acr_values` and `max_age` authorization request parameters are both **OPTIONAL** parameters defined in Section 3.1.2.1. of [OIDC]. This document does not introduce any changes in the authorization server behavior defined in [OIDC] for processing those parameters; hence, any authorization server implementing OpenID Connect will be able to participate in the flow described here with little or no changes. See Section 5 for more details.

The example authorization request URI below, which might be used after receiving the challenge in Figure 2, indicates to the authorization server that the client would like the authentication to occur according to the authentication context class reference identified by `myACR`.

```
https://as.example.net/authorize?client_id=s6BhdRkqt3
&response_type=code&scope=purchase&acr_values=myACR
```

Figure 4: Authorization Request Indicating `acr_values`

After the challenge in Figure 3, a client might direct the user agent to the following example authorization request URI where the `max_age` parameter indicates to the authorization server that the user-authentication event needs to have occurred no more than five seconds prior.

```
https://as.example.net/authorize?client_id=s6BhdRkqt3
&response_type=code&scope=purchase&max_age=5
```

Figure 5: Authorization Request Indicating `max_age`

5. Authorization Response

Section 5.5.1.1 of [OIDC] establishes that an authorization server receiving a request containing the `acr_values` parameter **MAY** attempt to authenticate the user in a manner that satisfies the requested authentication context class reference and include the corresponding value in the `acr` claim in the resulting ID Token. The same section also establishes that, in case the desired authentication level cannot be met, the authorization server **SHOULD** include a value reflecting the authentication level of the current session (if any) in the `acr` claim. Furthermore, Section 3.1.2.1 [OIDC] states that if a request includes the `max_age` parameter, the authorization server **MUST** include the `auth_time` claim in the issued ID Token. An authorization server complying with this specification will react to the presence of the `acr_values` and `max_age` parameters by including `acr` and `auth_time` in the access token (see Section 6 for details). Although [OIDC] leaves the authorization server free to decide how to handle the inclusion of `acr` in the ID Token when requested via `acr_values`, when it comes to access tokens in this specification, the authorization server **SHOULD** consider the requested acr value as necessary for successfully fulfilling the request. That is, the requested `acr` value is included in the access token if the authentication operation successfully met its requirements; otherwise, the authorization request fails and returns an `unmet_authentication_requirements` error as defined in [OIDCUAR]. The recommended behavior will help prevent clients getting stuck in a loop where the authorization server keeps returning tokens that the resource server already identified as not meeting its requirements.

6. Authentication Information Conveyed via Access Token

To evaluate whether an access token meets the protected resource's requirements, the resource server needs a way of accessing information about the authentication event by which that access token was obtained. This specification provides guidance on how to convey that information in conjunction with two common access-token-validation methods:

- the one described in [RFC9068], where the access token is encoded in JWT format and verified via a set of validation rules, and
- the one described in [RFC7662], where the token is validated and decoded by sending it to an introspection endpoint.

Authorization servers and resource servers **MAY** elect to use other encoding and validation methods; however, those are out of scope for this document.

6.1. JWT Access Tokens

When access tokens are represented as JSON Web Tokens (JWTs) [RFC7519], the `auth_time` and `acr` claims (per Section 2.2.1 of [RFC9068]) are used to convey the time and context of the user-authentication event that the authentication server performed during the course of obtaining the access token. It is useful to bear in mind that the values of those two parameters are established at user-authentication time and will not change in the event of access token renewals. See the aforementioned Section 2.2.1 of [RFC9068] for details. The following is a conceptual example showing the decoded content of such a JWT access token.

```
Header:

{"typ":"at+JWT","alg":"ES256","kid":"LTacESbw"}

Claims:

{
 "iss": "https://as.example.net",
 "sub": "someone@example.net",
 "aud": "https://rs.example.com",
 "exp": 1646343000,
 "iat": 1646340200,
 "jti" : "e1j3V_bKic8-LAEB_lccD0G",
 "client_id": "s6BhdRkqt3",
 "scope": "purchase",
 "auth_time": 1646340198,
 "acr": "myACR"
}
```

Figure 6: Decoded JWT Access Token

6.2. OAuth 2.0 Token Introspection

"OAuth 2.0 Token Introspection" [RFC7662] defines a method for a protected resource to query an authorization server about the active state of an access token as well as to determine metainformation about the token. The following two top-level introspection response members are defined to convey information about the user-authentication event that the authentication server performed during the course of obtaining the access token.

`acr`: String specifying an authentication context class reference value that identifies the authentication context class that was satisfied by the user-authentication event performed.

`auth_time`: Time when the user authentication occurred. A JSON numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the date/time of the authentication event.

The following example shows an introspection response with information about the user-authentication event by which the access token was obtained.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active": true,
  "client_id": "s6BhdRkqt3",
  "scope": "purchase",
  "sub": "someone@example.net",
  "aud": "https://rs.example.com",
  "iss": "https://as.example.net",
  "exp": 1639528912,
  "iat": 1618354090,
  "auth_time": 1646340198,
  "acr": "myACR"
}
```

Figure 7: Introspection Response

7. Authorization Server Metadata

Authorization servers can advertise their support of this specification by including in their metadata document, as defined in [RFC8414], the value `acr_values_supported`, as defined in Section 3 of [OIDCDISC]. The presence of `acr_values_supported` in the authorization server metadata document signals that the authorization server will understand and honor the `acr_values` and `max_age` parameters in incoming authorization requests.

8. Deployment Considerations

This specification facilitates the communication of requirements from a resource server to a client, which, in turn, can enable a smooth step up authentication experience. However, it is important to realize that the user experience achievable in every specific deployment is a function of the policies each resource server and authorization server pair establishes. Imposing constraints on those policies is out of scope for this specification; hence, it is perfectly possible for resource servers and authorization servers to impose requirements that are impossible for users to comply with or that lead to an undesirable user-experience outcome. The authentication prompts presented by the authorization server as a result of the method of propagating authentication requirements described here might require the user to perform some specific actions such as using multiple devices, having access to devices complying with specific security requirements, and so on. Those extra requirements, that are more concerned with how to comply with a particular requirement rather than indicating the identifier of the requirement itself, are out of scope for this specification.

9. Security Considerations

This specification adds to previously defined OAuth mechanisms. Their respective security considerations apply:

- OAuth 2.0 [RFC6749],
- JWT access tokens [RFC9068],
- Bearer `WWW-Authenticate` [RFC6750],
- token introspection [RFC7662], and
- authorization server metadata [RFC8414].

This document **MUST NOT** be used to position OAuth as an authentication protocol. For the purposes of this specification, the way in which a user authenticated with the authorization server to obtain an access token is salient information, as a resource server might decide whether to grant access on the basis of how that authentication operation was performed. Nonetheless, this specification does not attempt to define the mechanics by which authentication takes place, relying on a separate authentication layer to take care of the details. In line with other specifications of the OAuth family, this document assumes the existence of a session without going into the details of how it is established or maintained, what protocols are used to implement that layer (e.g., OpenID Connect), and so forth. Depending on the policies adopted by the resource server, the `acr_values` parameter introduced in Section 3 might unintentionally disclose information about the authenticated user, the resource itself, the authorization server, and any other context-specific data that an attacker might use to gain knowledge about their target. For example, a resource server requesting an acr value corresponding to a high level of assurance for some users but not others might identify possible high-privilege users to target with spearhead phishing attacks. Implementers should use care in determining what to disclose in the challenge and in what circumstances. The logic examining the incoming access token to determine whether or not a challenge should be returned can be executed either before or after the conventional token-validation logic, be it based on JWT validation, introspection, or any other method. The resource server **MAY** return a challenge without verifying the client presented a valid token. However, this approach will leak the required properties of an authorization token to an actor who has not proven they can obtain a token for this resource server.

As this specification provides a mechanism for the resource server to trigger user interaction, it's important for the authorization server and clients to consider that a malicious resource server might abuse that feature.

10. IANA Considerations

10.1. OAuth Extensions Error Registration

This specification registers the following error value in the "OAuth Extensions Error Registry" [IANA.OAuth.Params] established by [RFC6749].

Name: `insufficient_user_authentication`
Usage Location: resource access error response
Protocol Extension: OAuth 2.0 Step Up Authentication Challenge Protocol
Change controller: IETF
Specification document(s): Section 3 of RFC 9470

10.2. OAuth Token Introspection Response Registration

This specification registers the following values in the "OAuth Token Introspection Response" registry [IANA.OAuth.Params] established by [RFC7662].

Authentication Context Class Reference:

Name: `acr`
Description: Authentication Context Class Reference
Change Controller: IETF
Specification Document(s): Section 6.2 of RFC 9470

Authentication Time:

Name: `auth_time`
Description: Time when the user authentication occurred
Change Controller: IETF
Specification Document(s): Section 6.2 of RFC 9470

11. References

11.1. Normative References

[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>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[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>.

11.2. Informative References

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

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OIDCDISC] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[OIDCUAR] Lodderstedt, T., "OpenID Connect Core Error Code unmet_authentication_requirements", 8 May 2019, <https://openid.net/specs/openid-connect-unmet-authentication-requirements-1_0.html>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[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>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession at the Application Layer (DPoP)", RFC 9449, DOI 10.17487/RFC9449, August 2023, <https://www.rfc-editor.org/info/rfc9449>.

Acknowledgements

I wanted to thank the Academy, the viewers at home, the shampoo manufacturers, etc.

This specification was developed within the OAuth Working Group under the chairpersonship of Rifaat Shekh-Yusef and Hannes Tschofenig with Paul Wouters and Roman Danyliw serving as Security Area Directors. Additionally, the following individuals contributed ideas, feedback, corrections, and wording that helped shape this specification: Caleb Baker, Ivan Kanakarakis, Pieter Kasselman, Aaron Parecki, Denis Pinkas, Dima Postnikov, and Filip Skokan.

Some early discussion of the motivations and concepts that precipitated the initial draft version of this document occurred at the 2021 OAuth Security Workshop. The authors thank the organizers of the workshop (Guido Schmitz, Steinar Noem, and Daniel Fett) for hosting an event that is conducive to collaboration and community input.

Authors' Addresses

Vittorio Bertocci
Auth0/Okta
Email: vittorio@auth0.com

Brian Campbell
Ping Identity
Email: bcampbell@pingidentity.com