Internet Engineering Task Force (IETF)
Request for Comments: 8292
Category: Standards Track
ISSN: 2070-1721

M. Thomson
Mozilla
P. Beverloo
Google

November 2017

Voluntary Application Server Identification (VAPID) for Web Push

Abstract

An application server can use the Voluntary Application Server Identification (VAPID) method described in this document to voluntarily identify itself to a push service. The "vapid" authentication scheme allows a client to include its identity in a signed token with requests that it makes. The signature can be used by the push service to attribute requests that are made by the same application server to a single entity. The identification information can allow the operator of a push service to contact the operator of the application server. The signature can be used to restrict the use of a push message subscription to a single application server.

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/rfc8292.

Table of Contents

1. Introduction

The Web Push protocol [RFC8030] describes how an application server is able to request that a push service deliver a push message to a user agent.

As a consequence of the expected deployment architecture, there is no basis for an application server to be known to a push service prior to requesting delivery of a push message. Requiring that the push service be able to authenticate application servers places an unwanted constraint on the interactions between user agents and application servers, who are the ultimate users of a push service. That constraint would also degrade the privacy-preserving properties the protocol provides. For these reasons, [RFC8030] does not define a mandatory system for authentication of application servers.

An unfortunate consequence of the design of [RFC8030] is that a push service is exposed to a greater risk of denial-of-service attacks. While requests from application servers can be indirectly attributed to user agents, this is not always efficient or even sufficient. Providing more information about the application server directly to a push service allows the push service to better distinguish between legitimate and bogus requests.

Additionally, the design of [RFC8030] relies on maintaining the secrecy of push message subscription URIs. Any application server in possession of a push message subscription URI is able to send messages to the user agent. If use of a subscription could be limited to a single application server, this would reduce the impact of the push message subscription URI being learned by an unauthorized party.

1.1. Voluntary Identification

This document describes a system whereby an application server can volunteer information about itself to a push service. At a minimum, this provides a stable identity for the application server, though this could also include contact information, such as an email address.

A consistent identity can be used by a push service to establish behavioral expectations for an application server. Significant deviations from an established norm can then be used to trigger exception-handling procedures.

Voluntarily provided contact information can be used to contact an application server operator in the case of exceptional situations.

Experience with push service deployment has shown that software errors or unusual circumstances can cause large increases in push message volume. Contacting the operator of the application server has proven to be valuable.

Even in the absence of usable contact information, an application server that has a well-established reputation might be given preference over an unidentified application server when choosing whether to discard a push message.

1.2. Notational Conventions

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.

The terms "push message", "push service", "push message subscription", "application server", and "user agent" are used as defined in [RFC8030].

2. Application Server Self-Identification

Application servers that wish to self-identify generate and maintain a signing key pair. This key pair MUST be usable with the Elliptic Curve Digital Signature Algorithm (ECDSA) over the P-256 curve [FIPS186]. Use of this key when sending push messages establishes an identity for the application server that is consistent across multiple messages.

When requesting delivery of a push message, the application includes a JSON Web Token (JWT) [RFC7519], signed using its signing key. The token includes a number of claims as follows:

- An "aud" (Audience) claim in the token MUST include the Unicode serialization of the origin (Section 6.1 of [RFC6454]) of the push resource URL. This binds the token to a specific push service and ensures that the token is reusable for all push resource URLs that share the same origin.

- An "exp" (Expiry) claim MUST be included with the time after which the token expires. This limits the time over which a token is valid. An "exp" claim MUST NOT be more than 24 hours from the

time of the request. Limiting this to 24 hours balances the need for reuse against the potential cost and likelihood of theft of a valid token.

This JWT is included in an Authorization header field, using an authentication scheme of "vapid". A push service MAY reject a request with a 403 (Forbidden) status code [RFC7231] if the JWT signature or its claims are invalid. A push service MUST NOT use information from an invalid token.

The JWT MUST use a JSON Web Signature (JWS) [RFC7515]. The signature MUST use ECDSA on the NIST P-256 curve [FIPS186], which is identified as "ES256" [RFC7518].

2.1. Application Server Contact Information

If the application server wishes to provide contact details, it MAY include a "sub" (Subject) claim in the JWT. The "sub" claim SHOULD include a contact URI for the application server as either a "mailto:" (email) [RFC6068] or an "https:" [RFC2818] URI.

2.2. Additional Claims

An application server MAY include additional claims using public or private names (see Sections 4.2 and 4.3 of [RFC7519]). Since the JWT is in a header field, the size of additional claims SHOULD be kept as small as possible.

2.3. Cryptographic Agility

The "vapid" HTTP authentication scheme (Section 3) is used to identify the specific profile of JWT defined in this document. A different authentication scheme is needed to update the signature algorithm or other parameters. This ensures that existing mechanisms for negotiating authentication schemes can be used rather than defining new parameter negotiation mechanisms.

2.4. Example

An application server requests the delivery of a push message as described in [RFC8030]. If the application server wishes to self-identify, it includes an Authorization header field with credentials that use the "vapid" authentication scheme.

```
POST /p/JzLQ3raZJfFBR0aqvOMsLrt54w4rJUsV HTTP/1.1
Host: push.example.net
TTL: 30
Content-Length: 136
Content-Encoding: aes128gcm
Authorization: vapid
   t=eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9.eyJhdWQiOiJodHRwczovL3
     B1c2guZXhhbXBsZS5uZXQiLCJleHAiOjE0NTM1MjM3NjgsInN1YiI6Im1ha
     Wx0bzpwdXNoQGV4YW1wbGUuY29tIn0.i3CYb7t4xfxCDquptFOepC9GAu_H
     LGkMlMuCGSK2rpiUfnK9ojFwDXb1JrErtmysazNjjvW2L9OkSSHzvoD1oA,
   k=BA1Hxzyi1RUM1b5wjxsn7nGxAszw2u61m164i3MrAIxHF6YK5h4SDYic-dR
     uU_RCPCfA5aq9ojSwk5Y2EmClBPs
{ encrypted push message }
```

Figure 1: Requesting Push Message Delivery with JWT

Note that the example header fields in this document include extra line wrapping to meet formatting constraints.

The "t" parameter of the Authorization header field contains a JWT; the "k" parameter includes the base64url-encoded key that signed that token. The JWT input values and the JSON Web Key (JWK) [RFC7517] corresponding to the signing key are shown in Figure 2 with additional whitespace added for readability purposes. This JWT would be valid until 2016-01-23T04:36:08Z.

```
JWT header = { "typ": "JWT", "alg": "ES256" }
JWT body = { "aud": "https://push.example.net",
             "exp": 1453523768,
             "sub": "mailto:push@example.com" }
JWK = { "crv":"P-256",
        "kty":"EC",
        "x":"DUfHPKLVFQzVvnCPGyfucbECzPDa7rWbXriLcysAjEc",
        "y":"F6YK5h4SDYic-dRuU_RCPCfA5aq9ojSwk5Y2EmClBPs" }
```

Figure 2: Decoded Example Values

3. VAPID Authentication Scheme

This document defines a new HTTP authentication scheme [RFC7235] named "vapid". This authentication scheme carries a signed JWT, as described in Section 2, plus the key that signed that JWT.

This authentication scheme is for origin-server authentication only. Therefore, this authentication scheme MUST NOT be used with the Proxy-Authenticate or Proxy-Authorization header fields.

The challenge for the "vapid" authentication scheme contains only the "auth-scheme" production. No parameters are currently defined.

Two parameters are defined for this authentication scheme: "t" and "k". All unknown or unsupported parameters to "vapid" authentication credentials MUST be ignored. The "realm" parameter is ignored for this authentication scheme.

This authentication scheme is intended for use by an application server when using the Web Push protocol [RFC8030].

3.1. Token Parameter ("t")

The "t" parameter of the "vapid" authentication scheme carries a JWT as described in Section 2.

3.2. Public Key Parameter ("k")

In order for the push service to be able to validate the JWT, it needs to learn the public key of the application server. A "k" parameter is defined for the "vapid" authentication scheme to carry this information.

The "k" parameter includes an ECDSA public key [FIPS186] in uncompressed form [X9.62] that is encoded using base64url encoding [RFC7515].

Note: X9.62 encoding is used over JWK [RFC7517] for two reasons. A JWK does not have a canonical form, so X9.62 encoding makes it easier for the push service to handle comparison of keys from different sources. Secondarily, the X9.62 encoding is also considerably smaller.

Some elliptic curve implementations permit the same P-256 key to be used for signing and key exchange. An application server MUST select a different private key for the key exchange [RFC8291] and signing the authentication token. Though a push service is not obligated to check either parameter for every push message, a push service SHOULD reject push messages that have identical values for these parameters with a 400 (Bad Request) status code.

4. Subscription Restriction

The public key of the application server serves as a stable identifier for the server. This key can be used to restrict a push message subscription to a specific application server.

Subscription restriction reduces the reliance on endpoint secrecy by requiring that an application server provide a signed token when requesting delivery of a push message. This provides an additional level of protection against leaking of the details of the push message subscription.

4.1. Creating a Restricted Push Message Subscription

A user agent that wishes to create a restricted subscription includes the public key of the application server when requesting the creation of a push message subscription. This restricts use of the resulting subscription to application servers that are able to provide a valid JWT signed by the corresponding private key.

The user agent then adds the public key to the request to create a push message subscription. The push message subscription request is extended to include a body. The body of the request is a JSON object as described in [RFC7159]. The user agent adds a "vapid" member to this JSON object that contains a public key on the P-256 curve, encoded in the uncompressed form [X9.62] and base64url encoded [RFC7515]. The media type of the body is set to "application/webpush-options+json" (see Section 6.3 for registration of this media type).

A push service MUST ignore the body of a request that uses a different media type. For the "application/webpush-options+json" media type, a push service MUST ignore any members on this object that it does not understand.

The example in Figure 3 shows a restriction to the key used in Figure 1. Extra whitespace is added to meet formatting constraints.

```
POST /subscribe/ HTTP/1.1
Host: push.example.net
Content-Type: application/webpush-options+json
Content-Length: 104
{ "vapid": "BA1Hxzyi1RUM1b5wjxsn7nGxAszw2u61m164i3MrAIxH
            F6YK5h4SDYic-dRuU_RCPCfA5aq9ojSwk5Y2EmClBPs" }
```

Figure 3: Example Subscribe Request

An application might use the Push API [API] to provide the user agent with a public key.

4.2. Using Restricted Subscriptions

When a push message subscription has been restricted to an application server, the request for push message delivery MUST include a JWT signed by the private key that corresponds to the public key used when creating the subscription.

A push service MUST reject a message sent to a restricted push message subscription if that message includes no "vapid" authentication or invalid "vapid" authentication. A 401 (Unauthorized) status code might be used if the authentication is absent; a 403 (Forbidden) status code might be used if authentication is invalid.

"vapid" authentication is invalid if:

- either the authentication token or public key is not included in the request,
- the signature on the JWT cannot be successfully verified using the included public key,
- the current time is later than the time identified in the "exp" (Expiry) claim or more than 24 hours before the expiry time,
- the origin of the push resource is not included in the "aud" (Audience) claim, or
- the public key used to sign the JWT doesn't match the one that was included in the creation of the push message subscription.

A push service MUST NOT forward the JWT or public key to the user agent when delivering the push message.

An application server that needs to replace its signing key needs to request the creation of a new subscription by the user agent that is restricted to the updated key. Application servers need to remember the key that was used when requesting the creation of a subscription.

5. Security Considerations

This authentication scheme is vulnerable to replay attacks if an attacker can acquire a valid JWT. Sending requests using HTTPS as required by [RFC8030] provides confidentiality. Additionally, applying narrow limits to the period over which a replayable token can be reused limits the potential value of a stolen token to an attacker and can increase the difficulty of stealing a token.

An application server might offer falsified contact information. The application server asserts its email address or contact URI without any evidence to support the claim. A push service operator cannot use the presence of unvalidated contact information as input to any security-critical decision-making process.

Validation of a signature on the JWT requires a non-trivial amount of computation. For something that might be used to identify legitimate requests under denial-of-service attack conditions, this is not ideal. Application servers are therefore encouraged to reuse tokens, which permits the push service to cache the results of signature validation.

An application server that changes its signing key breaks linkability between push messages that it sends under different keys. A push service that relies on a consistent identity for application servers might categorize requests made with new keys differently. Gradual migration to a new signing key reduces the chances that requests that use the new key will be categorized as abusive.

6. IANA Considerations

This document registers a new authentication scheme, a registry for parameters of that scheme, and a media type for push options.

6.1. VAPID Authentication Scheme Registration

This document registers the "vapid" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" established by [RFC7235].

Authentication Scheme Name: vapid

Pointer to specification text: Section 3 of this document

6.2. VAPID Authentication Scheme Parameters

This document creates a "VAPID Authentication Scheme Parameters" registry for parameters to the "vapid" authentication scheme. These parameters are defined for use in requests (in the Authorization header field) and for challenges (in the WWW-Authenticate header field). This registry is under the "Web Push Parameters" grouping. The registry operates on the "Specification Required" policy [RFC8126].

Registrations MUST include the following information:

Parameter Name: A name for the parameter, which conforms to the "token" grammar [RFC7230]

Purpose (optional): Brief text identifying the purpose of the parameter

Header Field(s): The header field(s) in which the parameter can be used

Specification: A link to the specification that defines the format and semantics of the parameter

This registry initially contains the following entries:

Parameter Name	Purpose	Header Field(s)	Specification
t	JWT authentication token	Authorization	[RFC8292], Section 3.1
k	signing key	Authorization	[RFC8292], Section 3.2

6.3. application/webpush-options+json Media Type Registration

This document registers the "application/webpush-options+json" media type in the "Media Types" registry following the process described in [RFC6838].

Type name: application

Subtype name: webpush-options+json

Required parameters: none

Optional parameters: none

Encoding considerations: binary (JSON is UTF-8-encoded text)

Security considerations: See [RFC7159] for security considerations specific to JSON.

Interoperability considerations: See [RFC7159] for interoperability considerations specific to JSON.

Published specification: [RFC8292]

Applications that use this media type: Web browsers, via the Web Push protocol [RFC8030]

Fragment identifier considerations: none

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): .json

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Martin Thomson (martin.thomson@gmail.com)

Intended usage: LIMITED USE

Restrictions on usage: For use with the Web Push protocol [RFC8030]

Author: See "Authors' Addresses" section of [RFC8292].

Change controller: Internet Engineering Task Force

7. References

7.1. Normative References

[FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013.

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010, <https://www.rfc-editor.org/info/rfc6068>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[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>.

[RFC8030] Thomson, M., Damaggio, E., and B. Raymor, Ed., "Generic Event Delivery Using HTTP Push", RFC 8030, DOI 10.17487/RFC8030, December 2016, <https://www.rfc-editor.org/info/rfc8030>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8291] Thomson, M., "Message Encryption for Web Push", RFC 8291, DOI 10.17487/RFC8291, November 2017, <http://www.rfc-editor.org/info/rfc8291>.

[X9.62] ANSI, "Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005.

7.2. Informative References

[API] Beverloo, P., Thomson, M., van Ouwerkerk, M., Sullivan, B., and É. Fullea, "Push API", October 2017, <https://www.w3.org/TR/push-api/>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

Acknowledgements

This document would have been much worse than it is if not for the contributions of Benjamin Bangert, JR Conlin, Chris Karlof, Costin Manolache, Adam Roach, and others.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Peter Beverloo
Google

Email: beverloo@google.com