SIPCORE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

C. Holmberg
Ericsson

October 30, 2017

Push Notification with the Session Initiation Protocol (SIP)
draft-holmberg-sipcore-sip-push-02

Abstract

This document describes how push notification mechanisms can be used to wake up suspended Session Initiation Protocol (SIP) User Agents (UAs), in order to be able to receive and generate SIP requests. The document defines new SIP URI parameters, that can be used in a SIP REGISTER request to provide push notification information from the SIP User Agent (UA) to the SIP entity (realized as a SIP proxy in this document) that will send a push request to the push server in order to trigger a push notification towards the SIP UA.

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 http://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 May 3, 2018.

Table of Contents

1. Introduction

In order to save resources (e.g, battery life) some devices and operating systems require suspended Session Initiation Protocol (SIP) User Agents (UAs) [RFC3261] to be woken up using a push notification service. Typically each operating system uses a dedicated push notification service. For example, Apple iOS devices use the Apple Push Notification service (APNs).

Due to the restriction above, applications can not be woken up by non-push notification traffic. This means that a suspended SIP UA will not be able to receive an incoming SIP request (e.g., a SIP INVITE request).

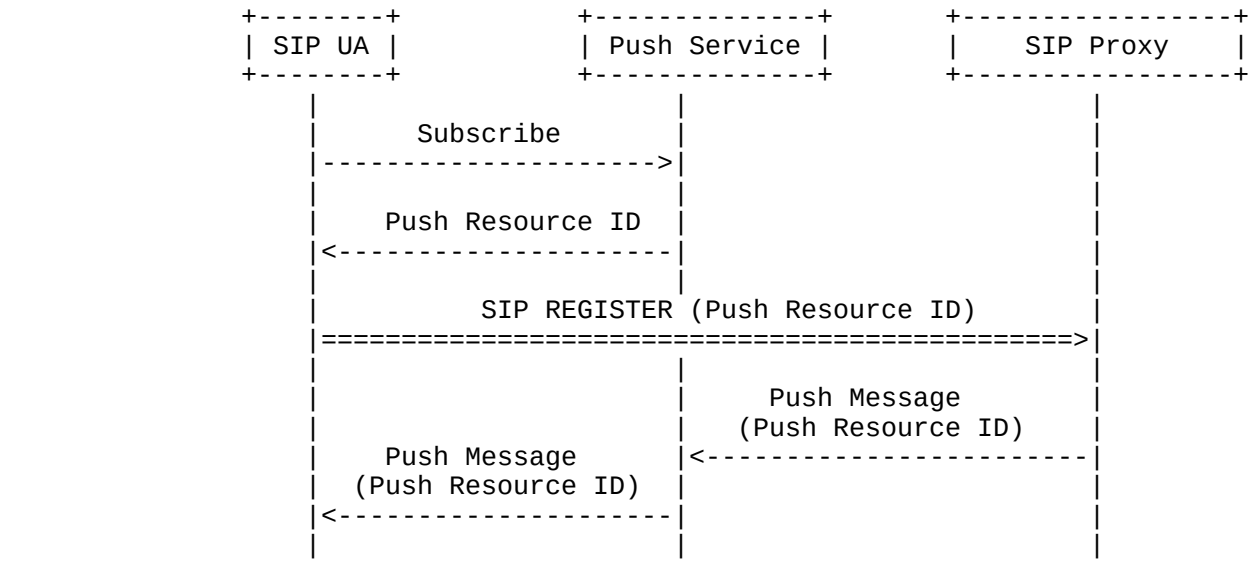
This document describes how push notification mechanisms can be used to wake up suspended SIP UAs, in order to be able to receive and generate SIP requests. The document defines new SIP URI parameters, that can be used in a SIP REGISTER request to provide push

notification information from the SIP UA to the SIP entity (realized as a SIP proxy in this document) that will send a push request to the push server in order to trigger a push notification towards the SIP UA.

When a SIP UA registers to a push service, it will receive a unique Push Resource ID (PRID) associated to that registration. The SIP UA will provide the PRID to the SIP network in a SIP REGISTER request. A SIP proxy (e.g., the SIP registrar) will store a mapping between the registered contact and the PRID.

When the SIP proxy receives a SIP request for a new session, or a stand-alone SIP request, addressed towards a SIP UA, the SIP proxy will send a push request to the push notification service used by the SIP UA, using the push resource ID associated with the registered contact of the SIP UA, in order to trigger a push notification towards the SIP UA. The SIP proxy will then forward the SIP request towards the SIP UA using normal SIP routing procedures. Once the SIP UA receives the push notification, it will be able to receive the SIP request (and generate a SIP request itself, if needed).

Different push notification mechanisms exist today. Some are based on there standardized mechanism defined in [RFC8030], while others are proprietary (e.g., the Apple Push Notification service). Figure 1 shows the generic push notification architecture supported by the mechanism in this document.




```
REGISTER sip:alice@example.com SIP/2.0
Via: SIP/2.0/TCP alicemobile.example.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Alice <sip:alice@example.com>
From: Alice <sip:alice@example.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:alice@alicemobile.example.com;
  pn-type=acme:acme-param;
  pn-prid="ZTY4ZDJlMzODE1NmUgKi0K">
Expires: 7200
Content-Length: 0
```

Figure 1: SIP Push Notification Architecture

2. Conventions

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

3. Push Resource ID (PRID)

When an entity registers with a Push Notification Server (PNS) is receives a unique Push Resource ID (PRID), which is a value associated with the registration.

The format of the PRID may vary depending on the PNS provider. The PRID may be part of a URI that can be used to retrieve the address and port of the PNS when sending push requests to the PNS. The PRID may also be a token value, in which case the address and port of the PNS needs to be provided using other means.

The details regarding discovery of the PNS, and the procedures for the push notification registration and maintenance are outside the scope of this document. The information needed to contact the PNS is typically pre-configured in the operating system (OS) of the device.

4. SIP User Agent (UA) Behavior

Once the SIP UA has registered with the PNS and received the PRID, and when the UA wants to receive push notifications triggered by the SIP proxy, the UA MUST send a SIP REGISTER using normal SIP registration procedures. The UA MUST add a pn-prid URI parameter and a pn-type URI parameter to the SIP Contact header field URI of the request. The pn-prid URI parameter contains the PRID value. The pn-type contains additional, PNS-specific, information.

As long as the UA wants the SIP proxy to continue sending push requests, the UA MUST include the pn-prid and pn-type URI parameters in every re-registration SIP REGISTER request sent towards the SIP proxy. Note that, in some cases, the PNS might update the PRID value, in which case the re-registration SIP REGISTER request will contain the new value.

If the UA at some point wants to stop the SIP proxy from sending push requests, the UA MUST send a SIP REGISTER request without the pn-prid and pn-type URI parameters.

If the UA expects to receive payload in the push notification, the UA MAY add a pn-enckey and a pn-encsec Contact header field URI parameter, in order to allow encryption of the data using the mechanism in [I-D.ietf-webpush-encryption]. The pn-enckey URI parameter contains the public key, and the pn-encsec URI parameter contains the authentication secret [I-D.ietf-webpush-encryption].

NOTE: End-to-end encryption of the payload between the SIP proxy and the SIP UA cannot be used if the push notification request payload

contains information that needs to be accessible by the push notification server.

5. SIP Proxy Behavior

When the SIP proxy receives a SIP request for a new dialog (e.g., a SIP INVITE request) or a non-dialog SIP request (e.g., a SIP MESSAGE request) aimed for a SIP UA, if the Request-URI of the request contains a pn-prid URI parameter, the SIP proxy triggers a push request towards the push notification server associated with the PRID. After that, the SIP proxy forwards the SIP request towards the SIP UA using normal SIP procedures.

The SIP proxy MUST NOT transport the SIP request as push request payload, instead of forwarding the request using normal SIP procedures.

In some cases the push notification provider can be retrieved from the pn-prid URI parameter. In other cases the pn-type URI parameter is used to identity the push notification provider.

If the proxy is not able to contact the push notification provider, or even determine which push notification provider to contact, it SHOULD reject the SIP request.

The protocol and format used for the push request depends on the push notification provider, and the details for constructing and sending the messages are outside the scope of this specification.

6. Network Address Translator (NAT) Considerations

Whenever the UA receives a push notification, if the SIP UA is located behind a Network Address Translator (NAT), the UA might need to take actions in order to establish a binding in the NAT, in order for an incoming SIP request to reach the UA. [RFC5626] and [RFC6223] define such mechanisms. This document does not require usage of a specific mechanism.

7. Grammar

The section defines new SIP URI parameters, by extending the grammar for "uri-parameter" as defined in [RFC3261]. The ABNF is as follows:

```
uri-parameter   =/ pn-prid / pn-type / pn-enccode / pn-enckey
pn-prid         = "pn-prid" EQUAL pvalue
pn-type         = "pn-type" EQUAL pns-provider COLON pns-param
pn-enccode      = "pn-enccode" EQUAL pvalue
pn-enckey       = "pn-enckey" EQUAL pvalue

pns-provider    = pvalue ; Colon (":") characters MUST be escaped
pns-param       = pvalue ; Colon (":") characters MUST be escaped

; pvalue as defined in RFC 3261
; EQUAL as defined in RFC 3261
; COLON as defined in RFC 3261
```

The format and semantics of pns-param is specific to a given pns-provider value.

8. PNS Registration Requirements

When a new value is registered to the PNS Sub-registry, a reference to a specification which describes the push notification service associated with the value is provided. That specification MUST contain the following information:

- How the values for the pn-prid parameter is retrieved and set by the SIP UA.
- The format of the pns-param part of the pns-type parameter, and how the value of the pns-param part is retrieved and set by the SIP UA.
- Whether there are any restrictions regarding usage of payload encryption [I-D.ietf-webpush-encryption] with the associated push notification service.

9. pn-prid and pn-type URI parameters for Apple Push Notification service

When the Apple Push Notification service (APNs) is used, the value of the pn-type URI parameter pns-provider parameter part is "apns". The pns-param part contains the APNs App ID, which is encoded by two values, separated by a period (.): Team ID and Bundle ID. The Team ID is provided by Apple and is unique to a development team. The Bundle ID is unique to a development team, and is a string that will can match a single application or a group of applications.

Example: pn-type = apns:DEF123GHIJ.com.yourcompany.yourexampleapp

When the Apple Push Notification service (APNs) is used, pn-type URI parameter pns-prid parameter part contains the device token, which is

a unique identifier assigned by Apple to a specific app on a specific device.

Example: pn-prid = 00fc13adff78512

For more information on the APNs App ID:

https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/AppID.html

For more information on the APNs device token:

https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW13

10. pn-prid and pn-type URI parameters for Google Firebase Cloud Messaging (FCM) push notification service

When Firebase Cloud Messaging (FCM) is used, the value of the pn-type URI parameter pns-provider parameter part is "fcm". The pns-param part contains the Sender ID.

When Firebase Cloud Messaging (FCM) is used, pn-type URI parameter pns-prid parameter part contains the Registration token, which generated by the FCM SDK for each client app instance.

For more information on the Sender ID and Registration token:

https://firebase.google.com/docs/cloud-messaging/concept-options

11. Security considerations

In addition to the information exchanged between a device and its PNS in order to establish a push notification subscription, the mechanism in this document does not require entities to provide any additional information to the PNS.

Push notification mechanisms provide different methods to ensure that malicious user cannot trigger push notifications to a device. Users of the mechanism in this document MUST take measures to prevent push notifications from being sent to a device from a malicious user.

In case entities do want to include payload in the push notifications, this document defines the means for using end-to-end payload encryption between the entity sending the push request and the entity receiving the associated push notification.

12. IANA considerations

This specification defines new SIP URI parameters that extend the registry created by [RFC3969]:

12.1. pn-prid

Parameter Name: pn-prid

Predefined Values: No

Reference: RFC XXXX

12.2. pn-type

Parameter Name: pn-type

Predefined Values: No

Reference: RFC XXXX

12.3. pn-enckey

Parameter Name: pn-enckey

Predefined Values: No

Reference: RFC XXXX

12.4. pn-enccode

Parameter Name: pn-enccode

Predefined Values: No

Reference: RFC XXXX

12.5. PNS Sub-registry Establishment

This section creates a new sub-registry, "PNS", under the sip-parameters registry: http://www.iana.org/assignments/sip-parameters.

The purpose of the sub-registry is to register SIP URI pn-type values.

This sub-registry is defined as a table that contains the following three columns:

Value: The token under registration

Description: The name of the push notification service

Document: A reference to the document defining the registration

This specification registers the following values:

Value	Description	Document
apns	Apple Push Notification service	[RFC XXXX]
fcm	Firebase Cloud Messaging	[RFC XXXX]

13. References

13.1. 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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3969] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, DOI 10.17487/RFC3969, December 2004, <https://www.rfc-editor.org/info/rfc3969>.

[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>.

13.2. Informative references

[RFC5626] Jennings, C., Ed., Mahy, R., Ed., and F. Audet, Ed., "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, DOI 10.17487/RFC5626, October 2009, <https://www.rfc-editor.org/info/rfc5626>.

[RFC6223] Holmberg, C., "Indication of Support for Keep-Alive", RFC 6223, DOI 10.17487/RFC6223, April 2011, <https://www.rfc-editor.org/info/rfc6223>.

[I-D.ietf-webpush-encryption] Thomson, M., "Message Encryption for Web Push", draft-ietf-webpush-encryption-09 (work in progress), September 2017.

Author’s Address

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

SIPCORE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 27, 2018

C. Holmberg
Ericsson

November 23, 2017

Push Notification with the Session Initiation Protocol (SIP)
draft-holmberg-sipcore-sip-push-03

Abstract

This document describes how push notification mechanisms can be used to wake up suspended Session Initiation Protocol (SIP) User Agents (UAs), in order to be able to receive and generate SIP requests. The document defines new SIP URI parameters, that can be used in a SIP REGISTER request to provide push notification information from the SIP User Agent (UA) to the SIP entity (realized as a SIP proxy in this document) that will send a push request to the push server in order to trigger a push notification towards the SIP UA.

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 http://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 May 27, 2018.

Table of Contents

1. Introduction

In order to save resources (e.g, battery life) some devices and operating systems require suspended Session Initiation Protocol (SIP) User Agents (UAs) [RFC3261] to be woken up using a push notification service. Typically each operating system uses a dedicated push notification service. For example, Apple iOS devices use the Apple Push Notification service (APNs).

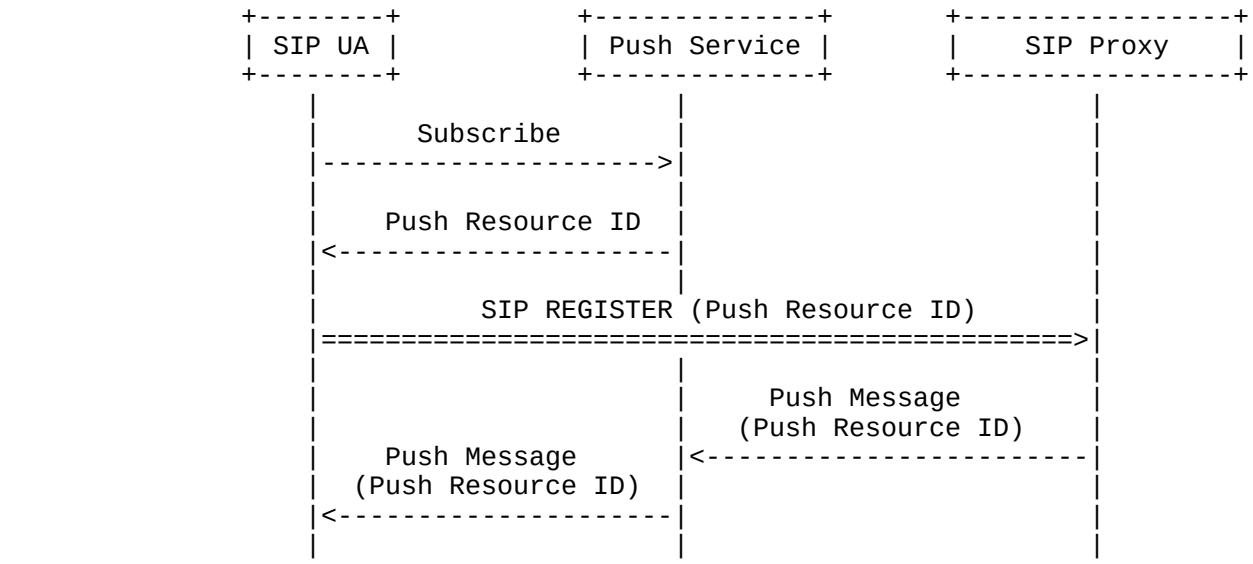
Due to the restriction above, applications can not be woken up by non-push notification traffic. This means that a suspended SIP UA will not be able to receive an incoming SIP request (e.g., a SIP INVITE request), or to send periodic re-registration requests.

This document describes how push notification mechanisms can be used to wake up suspended SIP UAs, in order to be able to receive and generate SIP requests. The document defines new SIP URI parameters, that can be used in a SIP REGISTER request to provide push notification information from the SIP UA to the SIP entity (realized as a SIP proxy in this document) that will send a push request to the push server in order to trigger a push notification towards the SIP UA.

When a SIP UA registers to a push service, it will receive a unique Push Resource ID (PRID) associated to that registration. The SIP UA will provide the PRID to the SIP network in a SIP REGISTER request. A SIP proxy (e.g., the SIP registrar) will store a mapping between the registered contact and the PRID.

When the SIP proxy receives a SIP request for a new session, or a stand-alone SIP request, addressed towards a SIP UA, or when the SIP proxy determines that the SIP UA needs to perform a re-registration, the SIP proxy will send a push request to the push notification service used by the SIP UA, using the push resource ID associated with the registered contact of the SIP UA, in order to trigger a push notification towards the SIP UA. The SIP proxy will then forward the SIP request towards the SIP UA using normal SIP routing procedures. Once the SIP UA receives the push notification, it will be to receive the SIP request, and to generate a SIP request (e.g., a SIP REGISTER) itself.

Different push notification mechanisms exist today. Some are based on there standardized mechanism defined in [RFC8030], while others are proprietary (e.g., the Apple Push Notification service). Figure 1 shows the generic push notification architecture supported by the mechanism in this document.




```
REGISTER sip:alice@example.com SIP/2.0
Via: SIP/2.0/TCP alicemobile.example.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Alice <sip:alice@example.com>
From: Alice <sip:alice@example.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:alice@alicemobile.example.com;
  pn-type=acme:acme-param;
  pn-prid="ZTY4ZDJlMzODE1NmUgKi0K">
Expires: 7200
Content-Length: 0
```

Figure 1: SIP Push Notification Architecture

2. Conventions

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

3. Push Resource ID (PRID)

When an entity registers with a Push Notification Server (PNS) is receives a unique Push Resource ID (PRID), which is a value associated with the registration.

The format of the PRID may vary depending on the PNS provider. The PRID may be part of a URI that can be used to retrieve the address and port of the PNS when sending push requests to the PNS. The PRID may also be a token value, in which case the address and port of the PNS needs to be provided using other means.

The details regarding discovery of the PNS, and the procedures for the push notification registration and maintenance are outside the scope of this document. The information needed to contact the PNS is typically pre-configured in the operating system (OS) of the device.

4. SIP User Agent (UA) Behavior

Once the SIP UA has registered with the PNS and received the PRID, and when the UA wants to receive push notifications triggered by the SIP proxy, the UA MUST send a SIP REGISTER using normal SIP registration procedures. The UA MUST add a pn-prid URI parameter and a pn-type URI parameter to the SIP Contact header field URI of the request. The pn-prid URI parameter contains the PRID value. The pn-type contains additional, PNS-specific, information.

When the SIP UA receives a push notification, it MUST perform a SIP re-registration [RFC3261] by sending a SIP REGISTER request. If there are Network Address Translators (NATs) between the SIP UA and the SIP proxy, the SIP REGISTER request will create NAT bindings allowing incoming SIP requests to reach the SIP UA. If the SIP proxy triggered the push notification because it wants to forward a SIP request towards the SIP UA, the receival of the SIP REGISTER request can be used by the SIP proxy as a trigger to forward the SIP request.

As long as the UA wants the SIP proxy to continue sending push requests, the UA MUST include the pn-prid and pn-type URI parameters in every re-registration SIP REGISTER request sent towards the SIP proxy. Note that, in some cases, the PNS might update the PRID value, in which case the re-registration SIP REGISTER request will contain the new value.

If the SIP UA at some point wants to stop the SIP proxy from sending push requests, the SIP UA MUST send a SIP REGISTER request without the pn-prid and pn-type URI parameters.

If the SIP UA expects to receive payload in the push notification, the SIP UA MAY add a pn-enckey and a pn-encsec Contact header field URI parameter, in order to allow encryption of the data using the mechanism in [I-D.ietf-webpush-encryption]. The pn-enckey URI parameter contains the public key, and the pn-encsec URI parameter contains the authentication secret [I-D.ietf-webpush-encryption].

NOTE: End-to-end encryption of the payload between the SIP proxy and the SIP UA cannot be used if the push notification request payload contains information that needs to be accessible by the push notification server.

5. SIP Proxy Behavior

5.1. Push Notification Provider Information

In some cases the push notification provider can be retrieved from the pn-prid URI parameter. In other cases the pn-type URI parameter is used to identity the push notification provider.

The protocol and format used for the push request depends on the push notification provider, and the details for constructing and sending the messages are outside the scope of this specification.

5.2. Trigger Periodic Re-registration

If the SIP UA needs to perform periodic re-registrations, the SIP proxy needs to have information about when those re-registrations are to be performed. The SIP proxy either needs to contain the SIP registrar functionality, or the SIP proxy needs to retreive the information from the registrar using some other mechanism.

When the SIP proxy receives an indication that the SIP UA needs to perform a re-registration, the SIP proxy triggers a push request towards the push notification server associated with the PRID.

5.3. SIP Request

When the SIP proxy receives a SIP request for a new dialog (e.g., a SIP INVITE request) or a non-dialog SIP request (e.g., a SIP MESSAGE request) aimed for a SIP UA, if the Request-URI of the request contains a pn-prid URI parameter, the SIP proxy triggers a push request towards the push notification server associated with the PRID. After that, the SIP proxy forwards the SIP request towards the SIP UA using normal SIP procedures.

As the push notification will trigger the SIP UA to perform a re-registration, the SIP proxy can use the receival of the SIP REGISTER request as a trigger to forward SIP request towards the SIP UA.

The SIP proxy MUST NOT transport the SIP request as push request payload, instead of forwarding the request using normal SIP procedures.

If the proxy is not able to contact the push notification provider, or even determine which push notification provider to contact, it SHOULD reject the SIP request.

If the SIP proxy is able to assume that the SIP UA is awake, and that the SIP UA is able to receive the SIP request, the SIP proxy MAY choose to not trigger a push notification request before trying to forward the SIP request towards the SIP UA. The SIP proxy can make such assumption e.g., if a TLS connection previously established by the SIP UA is still open.

6. Network Address Translator (NAT) Considerations

Whenever the UA receives a push notification, if the SIP UA is located behind a Network Address Translator (NAT), the UA might need to take actions in order to establish a binding in the NAT, in order for an incoming SIP request to reach the UA. By performing the re-registration the SIP UA will establish such NAT binding.

7. Grammar

The section defines new SIP URI parameters, by extending the grammar for "uri-parameter" as defined in [RFC3261]. The ABNF is as follows:

```
uri-parameter   =/ pn-prid / pn-type / pn-enccode / pn-enckey
pn-prid         = "pn-prid" EQUAL pvalue
pn-type         = "pn-type" EQUAL pns-provider COLON pns-param
pn-enccode      = "pn-enccode" EQUAL pvalue
pn-enckey       = "pn-enckey" EQUAL pvalue

pns-provider    = pvalue ; Colon (":") characters MUST be escaped
pns-param       = pvalue ; Colon (":") characters MUST be escaped

; pvalue as defined in RFC 3261
; EQUAL as defined in RFC 3261
; COLON as defined in RFC 3261
```

The format and semantics of pns-param is specific to a given pns-provider value.

8. PNS Registration Requirements

When a new value is registered to the PNS Sub-registry, a reference to a specification which describes the push notification service associated with the value is provided. That specification MUST contain the following information:

- How the values for the pn-prid parameter is retrieved and set by the SIP UA.
- The format of the pns-param part of the pns-type parameter, and how the value of the pns-param part is retrieved and set by the SIP UA.
- Whether there are any restrictions regarding usage of payload encryption [I-D.ietf-webpush-encryption] with the associated push notification service.

9. pn-prid and pn-type URI parameters for Apple Push Notification service

When the Apple Push Notification service (APNs) is used, the value of the pn-type URI parameter pns-provider parameter part is "apns". The pns-param part contains the APNs App ID, which is encoded by two values, separated by a period (.): Team ID and Bundle ID. The Team ID is provided by Apple and is unique to a development team. The Bundle ID is unique to a development team, and is a string that will can match a single application or a group of applications.

Example: pn-type = apns:DEF123GHIJ.com.yourcompany.yourexampleapp

When the Apple Push Notification service (APNs) is used, pn-type URI parameter pns-prid parameter part contains the device token, which is

a unique identifier assigned by Apple to a specific app on a specific device.

Example: pn-prid = 00fc13adff78512

For more information on the APNs App ID:

https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/AppID.html

For more information on the APNs device token:

https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW13

10. pn-prid and pn-type URI parameters for Google Firebase Cloud Messaging (FCM) push notification service

When Firebase Cloud Messaging (FCM) is used, the value of the pn-type URI parameter pns-provider parameter part is "fcm". The pns-param part contains the Sender ID.

When Firebase Cloud Messaging (FCM) is used, pn-type URI parameter pns-prid parameter part contains the Registration token, which generated by the FCM SDK for each client app instance.

For more information on the Sender ID and Registration token:

https://firebase.google.com/docs/cloud-messaging/concept-options

11. Security considerations

In addition to the information exchanged between a device and its PNS in order to establish a push notification subscription, the mechanism in this document does not require entities to provide any additional information to the PNS.

Push notification mechanisms provide different methods to ensure that malicious user cannot trigger push notifications to a device. Users of the mechanism in this document MUST take measures to prevent push notifications from being sent to a device from a malicious user.

In case entities do want to include payload in the push notifications, this document defines the means for using end-to-end payload encryption between the entity sending the push request and the entity receiving the associated push notification.

12. IANA considerations

This specification defines new SIP URI parameters that extend the registry created by [RFC3969]:

12.1. pn-prid

Parameter Name: pn-prid

Predefined Values: No

Reference: RFC XXXX

12.2. pn-type

Parameter Name: pn-type

Predefined Values: No

Reference: RFC XXXX

12.3. pn-enckey

Parameter Name: pn-enckey

Predefined Values: No

Reference: RFC XXXX

12.4. pn-enccode

Parameter Name: pn-enccode

Predefined Values: No

Reference: RFC XXXX

12.5. PNS Sub-registry Establishment

This section creates a new sub-registry, "PNS", under the sip-parameters registry: http://www.iana.org/assignments/sip-parameters.

The purpose of the sub-registry is to register SIP URI pn-type values.

This sub-registry is defined as a table that contains the following three columns:

Value: The token under registration

Description: The name of the push notification service

Document: A reference to the document defining the registration

This specification registers the following values:

Value	Description	Document
apns	Apple Push Notification service	[RFC XXXX]
fcm	Firebase Cloud Messaging	[RFC XXXX]

13. References

13.1. 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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3969] Camarillo, G., "The Internet Assigned Number Authority (IANA) Uniform Resource Identifier (URI) Parameter Registry for the Session Initiation Protocol (SIP)", BCP 99, RFC 3969, DOI 10.17487/RFC3969, December 2004, <https://www.rfc-editor.org/info/rfc3969>.

[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>.

13.2. Informative references

[I-D.ietf-webpush-encryption] Thomson, M., "Message Encryption for Web Push", draft-ietf-webpush-encryption-09 (work in progress), September 2017.

Author’s Address

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

SIP Core
Internet-Draft
Updates: 3261 (if approved)
Intended status: Standards Track
Expires: March 30, 2018

R. Shekh-Yusef, Ed.
Avaya
C. Holmberg
Ericsson
V. Pascual
webrtchacks

September 26, 2017

Third-Party Authentication for Session Initiation Protocol (SIP) draft-ietf-sipcore-sip-authn-01

Abstract

This document defines an authentication mechanism for SIP, that is based on the OAuth 2.0 and OpenID Connect Core 1.0 specifications, to enable the delegation of the user authentication to a dedicated third-party IdP entity that is separate from the SIP network elements that provide the SIP service.

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 March 30, 2018.

Table of Contents

1. Introduction

The SIP protocol [RFC3261] uses the framework used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

OAuth 2.0 [RFC6749] defines a token based authorization framework to allow clients to access resources on behalf of their user.

The OpenID Connect 1.0 [OPENID] specifications defines a simple identity layer on top of the OAuth 2.0 protocol, which enables clients to verify the identity of the user based on the authentication performed by a dedicated IdP entity, as well as to obtain basic profile information about the user.

This document defines an authentication mechanism for SIP, that is based on the OAuth 2.0 and OpenID Connect Core 1.0 specifications, to enable the delegation of the user authentication to a dedicated third-party IdP entity that is separate from the SIP network elements that provide the SIP service.

1.1. Terminology

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

1.2. Roles

resource owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

In a typical SIP network, it is the management element in the system that acts as a resource owner.

resource server

The server hosting the protected resources or services, capable of accepting and responding to protected resource and services requests using access tokens.

OAuth 2.0 client

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

SIP client

An application making requests to access SIP services on behalf of the end-user.

authorization server

The server issuing tokens to the OAuth 2.0 client or SIP Client after successfully authenticating the resource owner and obtaining authorization.

Identity Provider (IdP)

This definition is borrowed from [MITKB]

"IdP (Identity Provider), is a system that creates, maintains, and manages identity information for principals (users, services, or systems) and provides principal authentication to other service providers (applications) within a federation or distributed network. It is a trusted third party that can be relied upon by users and servers when users and servers are establishing a dialog that must be authenticated. The IdP sends an attribute assertion containing trusted information about the user to the SP".

1.3. ID Token

ID token, as defined in the OpenID document, is a security token that contains claims about the authentication of an end-user by an authorization server.

1.4. SIP User Agent Types

[RFC6749] defines two types of clients, confidential and public, that apply to the SIP User Agents.

- Confidential User Agent: is a SIP UA that is capable of maintaining the confidentiality of the user credentials and any tokens obtained using these user credentials.

- Public User Agent: is a SIP UA that is incapable of maintainings the confidentiality of the user credentials and any obtained tokens.

1.5. Authentication Types

There are two types of user authentications in SIP:

- Proxy-to-User: which allows a server that is providing a service to authenticate the identity of a user before providing the service.

- User-to-User: which allows a user receiving a request to authenticate the identity of the remote user before processing the request.

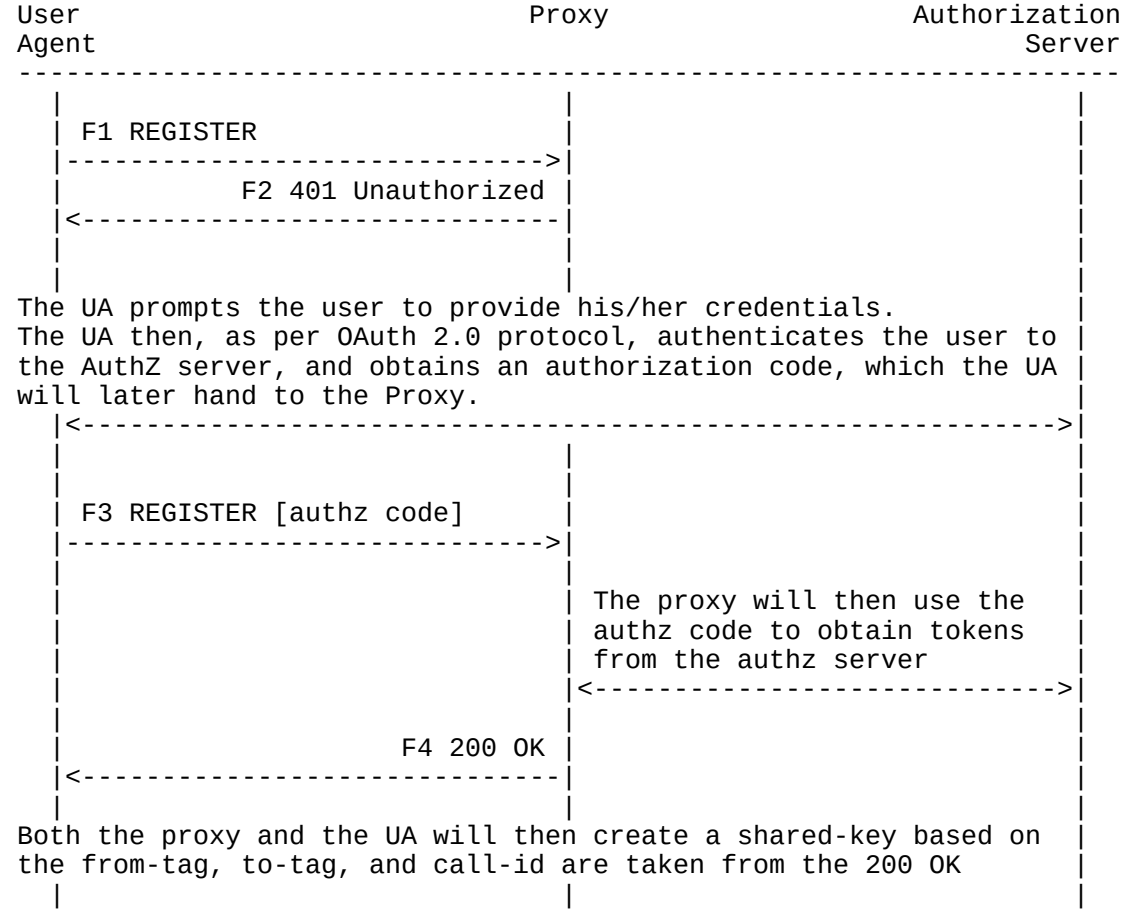
The mechanism defined in this document addresses the proxy-to-user authentication only. For user-to-user authentication refer to the mechanism defined in [STIR].

2. Authentication using the Authorization Code Flow

Authorization Code Flow is used by the SIP UA to authenticate to a third-party IdP entity to obtain an authorization code that would be later used by the SIP Proxy to obtain tokens to allow the SIP UA to register and get service from the SIP network.

2.1. Public UA with Rich UI

The following figure provides a high level view of flow of messages for the user authentication using a Public UA that has a rich UI that would prompt the user for his credentials:



The UA initially sends a REGISTER request (F1) without providing any credentials. The proxy redirects the UA by responding with 401 Unauthorized (F2).

The UA will then contact the Authorization Server and obtain an authorization code to be used with the SIP proxy.

The UA then retries the request (F3) and includes the authorization code in the body of the request.

The proxy then contacts the Authorization Server and exchanges the authorization code for tokens. If the proxy is successful in exchanging the authorization code with the tokens, the proxy then replies with 200 OK to complete the registration process, and locally generates the shared-key with the UA for this user.

When the UA receives the 200 OK, it will follow the same procedure used by the proxy and calculate its shared-key locally.

2.1.1. Registration

The UA initiates the process by sending a REGISTER request (F1) to the proxy. The proxy will redirect the UA to the Authorization Server by responding with 401 Unauthorized (F2) that includes the address of the Authorization Server in the form of an HTTP URI in a Location header field, as defined in RFC7231, section 7.1.2.

[[OPEN ISSUE]] The above text suggests defining a new Location header to carry the authorization server URL. Is that reasonable? other ideas?

The UA will then contact the Authorization Server and obtain an authorization code to be used with the SIP proxy. The method used by the UA to obtain the code is out of scope for this document.

The UA will then send a new REGISTER request (F3) and include the authorization code in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The proxy then contacts the Authorization Server and exchanges the authorization code for access token, refresh token, and maybe ID token. The method used by the UA to obtain the tokens is out of scope for this document.

If the proxy is successful in exchanging the authorization code with the tokens, the proxy then responds with 200 OK (F4) to the UA to complete the registration process.

2.1.2. Shared-Key

After sending the 200 OK to the UA to complete the registration process, the proxy and the UA use the HMAC-SHA256(key, message) to calculates the shared-key associated with this user as follows:

key

The authorization code obtained from the Authorization Server.

message

The concatenation of the ’from-tag’, ’to-tag’, and ’call-id’ of the 200 OK that completes the registration process.

This shared-key will be used to allow the UA to re-register to the proxy, in case of a connection loss to the proxy, without the need to obtain a new code or prompt the user for his credentials.

2.1.3. Re-Registration Requests

When the UA loses its connection to the proxy and it wants to send a new registration request to the proxy, the UA will send a new REGISTER request and include the proof-of-possession (pop) of the shared-key in the body of the request:

grant_type (REQUIRED)

Value MUST be set to "proof_of_possession".

pop (REQUIRED)

The pop calculated the first time the UA registered with the proxy.

The pop is calculated using the shared-key as follows:

```
pop = HMAC-SHA256(shared-key, digest-string)
```

See rfc4474, section 9, for the SIP headers to hash to create digest-string.

[[OPEN ISSUE]] Is there any issue with using digest-string as defined in RFC4474?

[[OPEN ISSUE]] Should pop not be limited to re-registration, and instead be used with all subsequent requests? If the answer is yes, a new header should be defined to carry the pop instead of carrying it in the payload.

2.1.4. Token Refresh

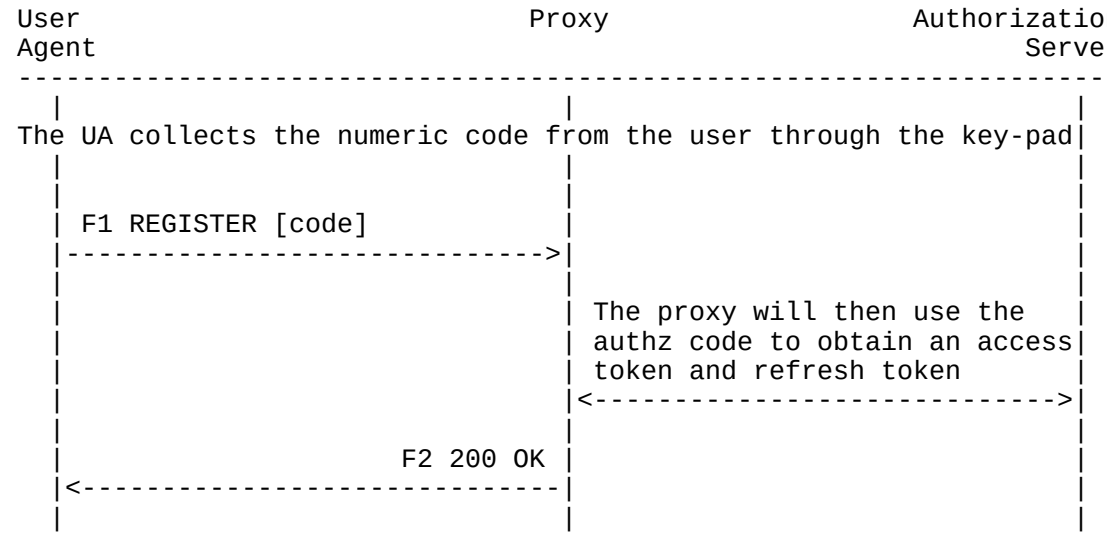
Before the tokens expire, the proxy makes a refresh request to the Authorization Server to try to obtain new tokens. The method used by the UA to refresh the tokens is out of scope for this document.

If the proxy fails to refresh the tokens, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process again.

2.2. Public UA with Limited UI

The following figure provides a high level view of flow of messages for the user authentication using a Public UA that has a limited UI that cannot prompt the user for his credentials.

This use case requires the user to use his browser to authenticate to the Authorization Server and obtain a short lived numeric authorization code that would be used by the phone to register with the SIP proxy.



2.2.1. Registration

The UA will send a REGISTER request (F1) and include the code in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The code received from the authorization server through the browser.

The proxy then contacts the Authorization Server and exchanges the authorization code for access token, refresh token, and maybe an ID token. The method used by the UA to obtain the tokens is out of scope for this document.

If the proxy is successful in exchanging the authorization code with the tokens, the proxy then responds with 200 OK (F2) to the UA to complete the registration process.

2.2.2. Shared-Key

After sending the 200 OK to the UA to complete the registration process, the proxy and the UA use the HMAC-SHA256(key, message) to calculates the shared-key associated with this user as follows:

key

The authorization code obtained from the Authorization Server.

message

The concatenation of the 'from-tag', 'to-tag', and 'call-id' of the 200 OK that completes the registration process.

This shared-key will be used to allow the UA to re-register to the proxy, in case of a connection loss to the proxy, without the need to obtain a new authorization code.

2.2.3. Token Refresh

Before the tokens expire, the proxy makes a refresh request to the Authorization Server to try to obtain new tokens. The method used by the UA to refresh the tokens is out of scope for this document.

If the proxy fails to refresh the tokens, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process again.

2.2.4. Re-Registration Requests

When the UA loses its connection to the proxy and it wants to send a new registration request to the proxy, the UA will send a new REGISTER request and include the proof-of-possession (pop) of the shared-key in the body of the request:

grant_type (REQUIRED)

Value MUST be set to "proof_of_possession".

pop (REQUIRED)

The pop calculated the first time the UA registered with the proxy.

The pop is calculated using the shared-key as follows:

pop = HMAC-SHA256(shared-key, digest-string)

See rfc4474, section 9, for the SIP headers to hash to create digest-string.

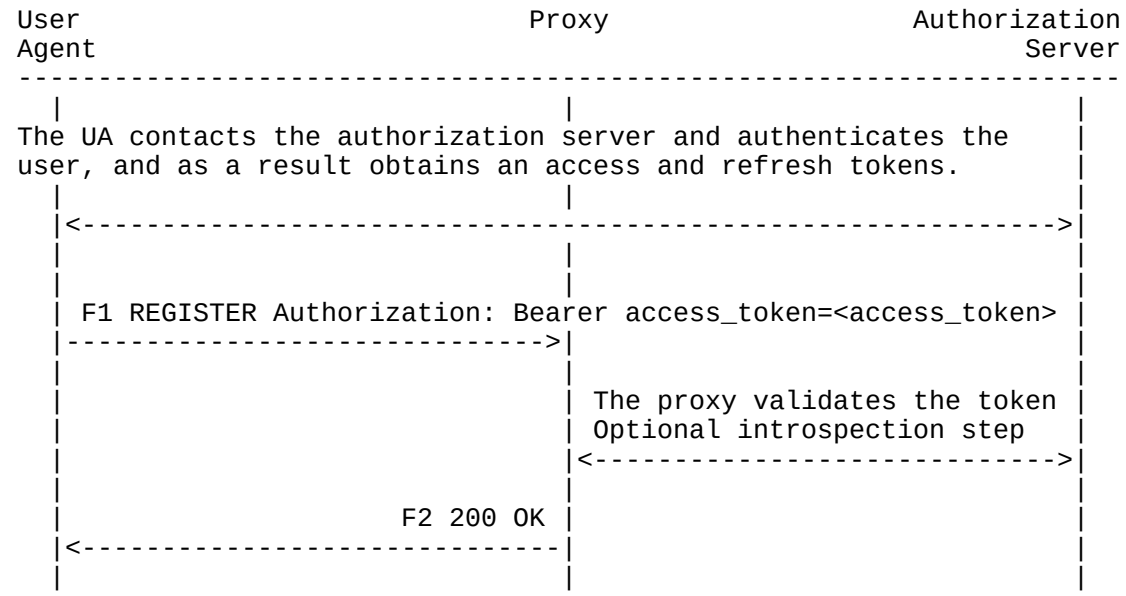
[[OPEN ISSUE]] Should this be not limited to re-registration, and instead be used with all subsequent requests?

3. Authentication using the Resource Owner Password Credentials flow

The resource owner password credentials flow is used by a Confidential UA with rich UI to authenticate to a third-party IdP entity and to directly obtain tokens to be able to register and get service from the SIP network.

3.1. Overview

The following figure provides a high level view of flow of messages for the OAuth Resource Owner Password Credentials flow:



3.2. Registration

The UA first contacts the Authorization Server to authenticate the user and obtain tokens to be used to get access to the SIP network. The method used by the UA to obtain the tokens is out of scope for this document.

The UA starts the registration process with the SIP proxy by sending a REGISTER request (F1) with the access token it obtained previously.

The UA includes an Authorization header field with the Bearer scheme in the request to carry the access token obtained previously.

The proxy then validates the token, and MAY perform an introspection step to get more information about the token and its scope. The introspection step is out of scope for this document.

When the proxy is satisfied with the token, it then replies with the 200 OK to complete the registration process.

3.3. Subsequent Requests

All subsequent requests from the UA MUST include a valid access token. The UA MUST obtain a new access token before the access token expiry period to continue to get service from the system.

4. Authorization Header Syntax

This section describes the syntax of the authorization header with the Bearer scheme.

```
Authorization = "Authorization" HCOLON "Bearer" LWS
                "access_token" EQUAL access_token
access_token = quoted-string
```

5. Security Considerations

<Security considerations text>

6. IANA Considerations

<IANA considerations text>

7. Acknowledgments

<Acknowledgments text>

8. Normative References

[MITKB] "IdP (Identity Provider)", MIT Knowledge Base, http://kb.mit.edu/confluence/x/XoK2, March 2011.

[OPENID] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", February 2014.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, H., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC7662] Richer, J., "OAuth 2.0 Token Introspection", RFC 7662, October 2015.

Authors' Addresses

Rifaat Shekh-Yusef (editor)
Avaya
250 Sidney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5176
EMail: rifaat.ietf@gmail.com

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: christer.holmberg@ericsson.com

Victor Pascual
webrtchacks
Spain

EMail: victor.pascual.avila@gmail.com

SIP Core
Internet-Draft
Updates: 3261 (if approved)
Intended status: Standards Track
Expires: August 9, 2018

R. Shekh-Yusef, Ed.
Avaya
C. Holmberg
Ericsson
V. Pascual
webrtchacks

February 5, 2018

Third-Party Authentication for Session Initiation Protocol (SIP) draft-ietf-sipcore-sip-authn-02

Abstract

This document defines an authentication mechanism for SIP, that is based on the OAuth 2.0 and OpenID Connect Core 1.0 specifications, to enable the delegation of the user authentication to a dedicated third-party IdP entity that is separate from the SIP network elements that provide the SIP service.

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 August 9, 2018.

Copyright Notice

Table of Contents

1. Introduction

The SIP protocol [RFC3261] uses the framework used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

OAuth 2.0 [RFC6749] defines a token based authorization framework to allow clients to access resources on behalf of their user.

The OpenID Connect 1.0 [OPENID] specifications defines a simple identity layer on top of the OAuth 2.0 protocol, which enables clients to verify the identity of the user based on the authentication performed by a dedicated IdP entity, as well as to obtain basic profile information about the user.

This document defines an authentication mechanism for SIP, that is based on the OAuth 2.0 and OpenID Connect Core 1.0 specifications, to enable the delegation of the user authentication to a dedicated third-party IdP entity that is separate from the SIP network elements that provide the SIP service.

1.1. Terminology

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

1.2. Roles

resource owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

In a typical SIP network, it is the management element in the system that acts as a resource owner.

resource server

The server hosting the protected resources or services, capable of accepting and responding to protected resource and services requests using access tokens.

OAuth 2.0 client

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

SIP client

An application making requests to access SIP services on behalf of the end-user.

authorization server

The server issuing tokens to the OAuth 2.0 client or SIP Client after successfully authenticating the resource owner and obtaining authorization.

Identity Provider (IdP)

This definition is borrowed from [MITKB]

"IdP (Identity Provider), is a system that creates, maintains, and manages identity information for principals (users, services, or systems) and provides principal authentication to other service providers (applications) within a federation or distributed network. It is a trusted third party that can be relied upon by users and servers when users and servers are establishing a dialog that must be authenticated. The IdP sends an attribute assertion containing trusted information about the user to the SP".

1.3. ID Token

ID token, as defined in the OpenID document, is a security token that contains claims about the authentication of an end-user by an authorization server.

1.4. SIP User Agent Types

[RFC6749] defines two types of clients, confidential and public, that apply to the SIP User Agents.

- Confidential User Agent: is a SIP UA that is capable of maintaining the confidentiality of the user credentials and any tokens obtained using these user credentials.

- Public User Agent: is a SIP UA that is incapable of maintainings the confidentiality of the user credentials and any obtained tokens.

1.5. Authentication Types

There are two types of user authentications in SIP:

- Proxy-to-User: which allows a server that is providing a service to authenticate the identity of a user before providing the service.

- User-to-User: which allows a user receiving a request to authenticate the identity of the remote user before processing the request.

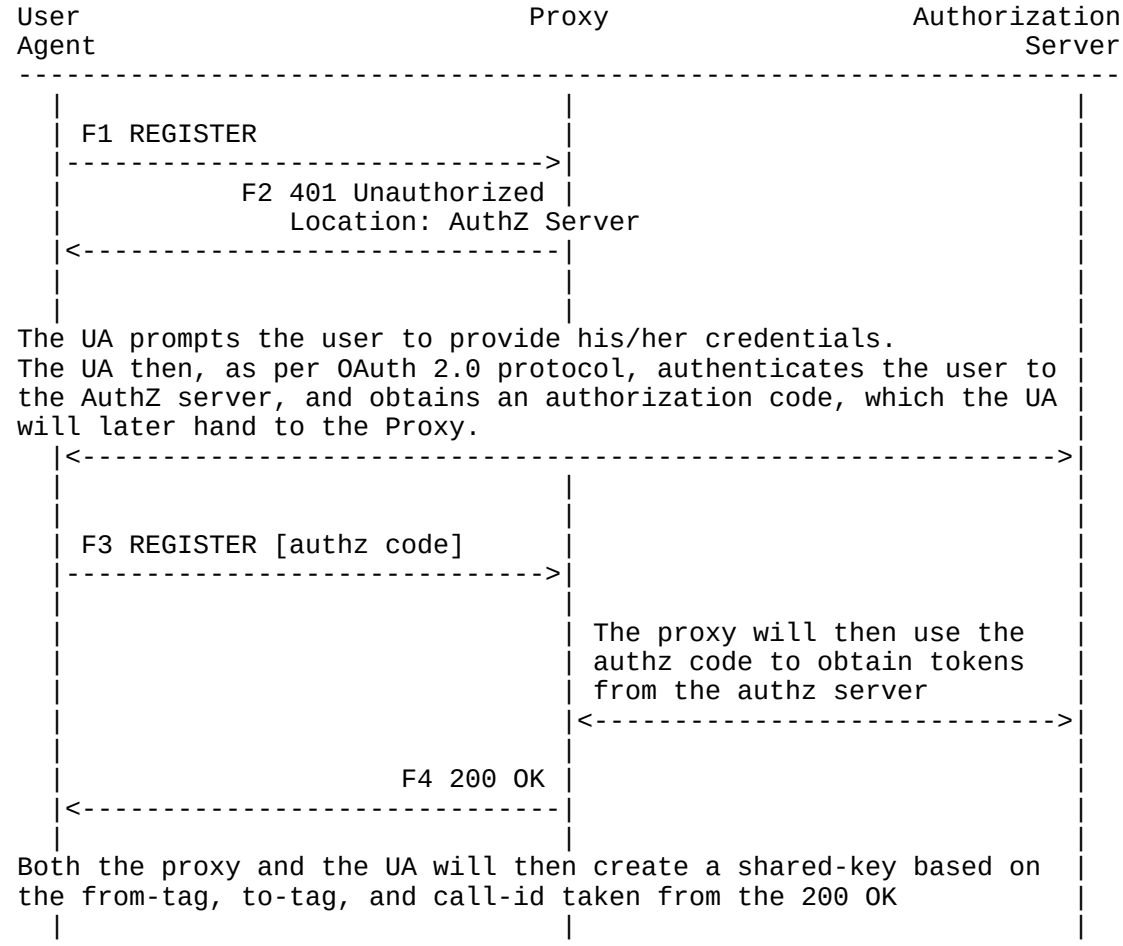
The mechanism defined in this document addresses the proxy-to-user authentication only. For user-to-user authentication refer to the mechanism defined in [RFC474bis].

2. Authentication using the Authorization Code Flow

Authorization Code Flow is used by the SIP UA to authenticate to a third-party IdP entity and to obtain an authorization code that would be later used by the SIP Proxy to obtain tokens to allow the SIP UA to register and get service from the SIP network.

2.1. Public UA with Rich UI

The following figure provides a high level view of flow of messages for the user authentication using a Public UA that has a rich UI that would prompt the user for his credentials:



The UA initially sends a REGISTER request (F1) without providing any credentials. The proxy redirects the UA by responding with 401 Unauthorized (F2).

The UA will then contact the Authorization Server and obtain an authorization code to be used with the SIP proxy.

The UA then retries the request (F3) and includes the authorization code in the body of the request.

The proxy then contacts the Authorization Server and exchanges the authorization code for tokens. If the proxy is successful in exchanging the authorization code with the tokens, the proxy then replies with 200 OK to complete the registration process, and locally generates the shared-key with the UA for this user.

When the UA receives the 200 OK, it will follow the same procedure used by the proxy and calculate its shared-key locally.

2.1.1. Initial Registration

The UA initiates the process by sending a REGISTER request (F1) to the proxy. The proxy will redirect the UA to the Authorization Server by responding with 401 Unauthorized (F2) that includes the address of the Authorization Server in the form of an HTTP URI in a Location header field, as defined in [RFC7231], section 7.1.2.

The UA will then contact the Authorization Server and obtain an authorization code to be used with the SIP proxy. The method used by the UA to obtain the code is out of scope for this document.

The UA will then send a new REGISTER request (F3) and include the authorization code, using the "application/x-www-form-urlencoded" format, in the body of the request with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The authorization code received from the authorization server.

The proxy then contacts the Authorization Server and exchanges the authorization code for access token, refresh token, and maybe ID

token. The method used by the UA to obtain the tokens is out of scope for this document.

If the proxy is successful in exchanging the authorization code with the tokens, the proxy then responds with 200 OK (F4) to the UA to complete the registration process; otherwise, the proxy MUST reply with 401 Unauthorized.

2.1.2. Shared-Key

The shared-key could be used to allow the UA to recover from a connection loss to the server without the need to prompt the user for credentials.

If the server supports the use of shared-key, it MUST indicate that by adding the new sip.shared-key parameter to the feature-caps header in the 200 OK response to the REGISTER request.

After sending the 200 OK to the UA to complete the registration process, assuming that both the server and the client support his feature, the proxy and the UA use the HMAC-SHA256(key, message) to calculates the shared-key associated with this user as follows:

key

The authorization code obtained from the Authorization Server.

message

The concatenation of the ’from-tag’, ’to-tag’, and ’call-id’ of the 200 OK that completes the registration process.

2.1.3. Subsequent Registration

When the UA loses its connection to the proxy and it wants to send a new registration request to the proxy, the UA will send a new REGISTER request and include the proof-of-possession (pop) of the shared-key in the body of the request, using the "application/x-www-form-urlencoded" format:

grant_type (REQUIRED)

Value MUST be set to "proof_of_possession".

pop (REQUIRED)

The pop calculated using the shared-key created the first time the UA registered with the proxy.

The pop is calculated using the shared-key as follows:

pop = HMAC-SHA256(shared-key, digest-string)

See [RFC4474], section 9, for the SIP headers to hash to create digest-string.

If the server supports the pop mechanism, then the server must validate the pop provided by the client. If the validation is successful, the server MUST reply with a 200 OK to complete the registration process; otherwise, the server MUST reply with 401 Unauthorized.

2.1.4. Token Refresh

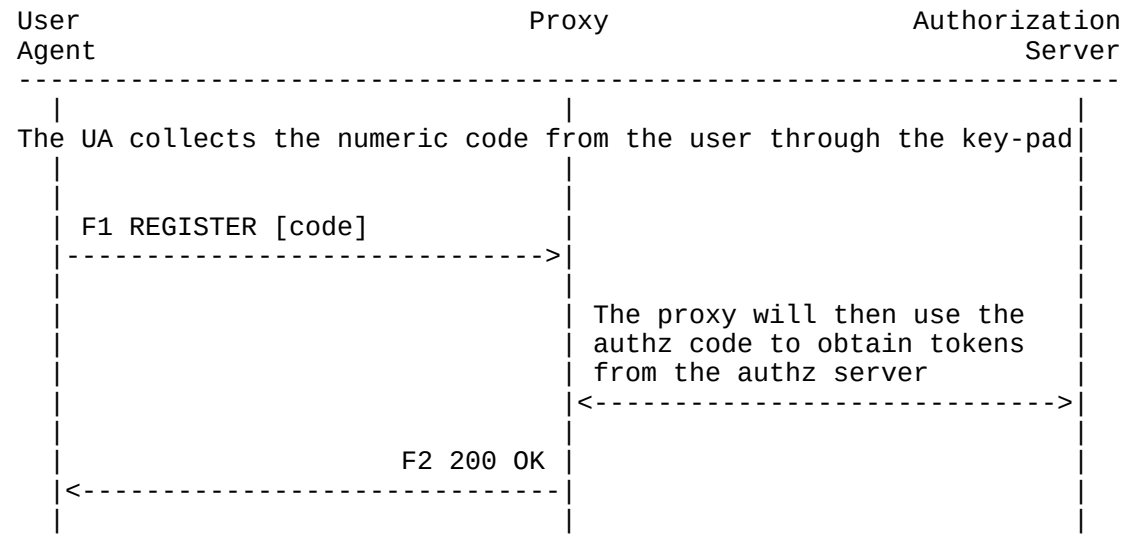
Before the tokens expire, the proxy makes a refresh request to the Authorization Server to try to obtain new tokens. The method used by the UA to refresh the tokens is out of scope for this document.

If the proxy fails to refresh the tokens, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process again.

2.2. Public UA with Limited UI

The following figure provides a high level view of flow of messages for the user authentication using a Public UA that has a limited UI that cannot prompt the user for his credentials.

This use case requires the user to use an out-of-band mechanism (e.g. a Browser or a One-Time-Password (OTP) application) to authenticate to the Authorization Server and obtain a short lived numeric authorization code that would be used by the phone to register with the SIP proxy.



2.2.1. Registration

The UA will send a REGISTER request (F1) and include the code in the body of the request, using the "application/x-www-form-urlencoded" format, with the following parameters:

grant_type (REQUIRED)

Value MUST be set to "authorization_code".

code (REQUIRED)

The code received from the authorization server through the out-of-bound mechanism.

The proxy then contacts the Authorization Server and exchanges the authorization code for access token, refresh token, and maybe an ID token. The method used by the UA to obtain the tokens is out of scope for this document.

If the proxy is successful in exchanging the authorization code with the tokens, the proxy then responds with 200 OK (F2) to the UA to complete the registration process; otherwise, the proxy MUST reply with 401 Unauthorized.

2.2.2. Shared-Key

The shared-key could be used to allow the UA to recover from a connection loss to the server without the need to prompt the user for credentials.

If the server supports the use of shared-key, it MUST indicate that by adding the new sip.shared-key parameter to the feature-caps header in the 200 OK response to the REGISTER request.

After sending the 200 OK to the UA to complete the registration process, assuming that both the server and the client support his feature, the proxy and the UA use the HMAC-SHA256(key, message) to calculates the shared-key associated with this user as follows:

key

The authorization code obtained from the Authorization Server.

message

The concatenation of the ’from-tag’, ’to-tag’, and ’call-id’ of the 200 OK that completes the registration process.

2.2.3. Token Refresh

Before the tokens expire, the proxy makes a refresh request to the Authorization Server to try to obtain new tokens. The method used by the UA to refresh the tokens is out of scope for this document.

If the proxy fails to refresh the tokens, then it MUST challenge the next request from the UA, and as a result the UA MUST go through the authorization process again.

2.2.4. Subsequent Registration

When the UA loses its connection to the proxy and it wants to send a new registration request to the proxy, the UA will send a new REGISTER request and include a proof-of-possession (pop) of the shared-key in the body of the request, using the "application/x-www-form-urlencoded" format:

grant_type (REQUIRED)

Value MUST be set to "proof_of_possession".

pop (REQUIRED)

The pop calculated using the shared-key created the first time the UA registered with the proxy.

The pop is calculated using the shared-key as follows:

pop = HMAC-SHA256(shared-key, digest-string)

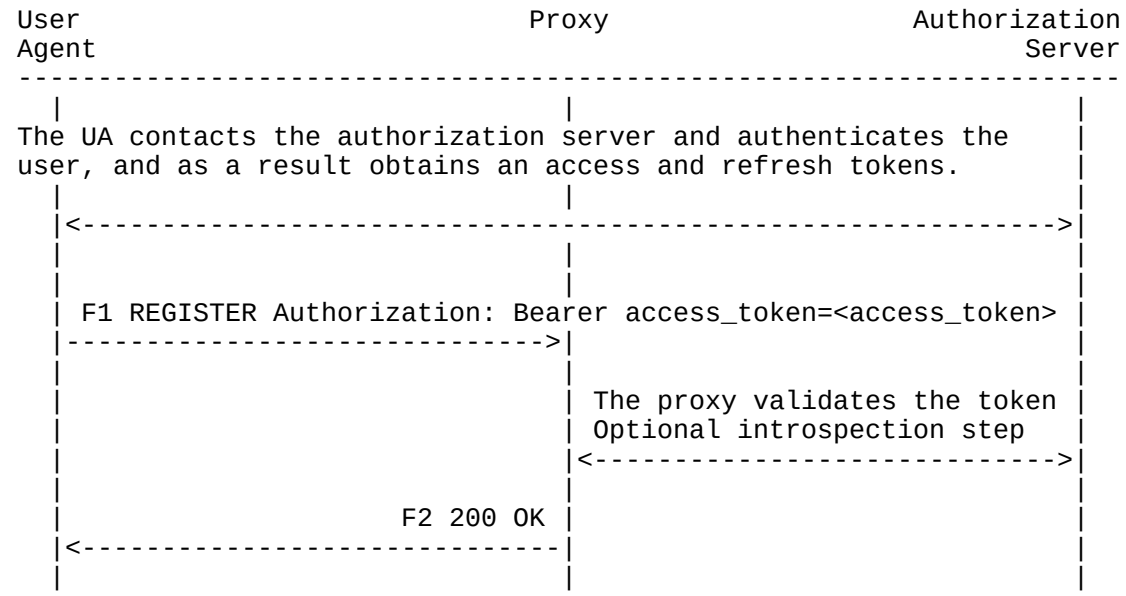
See [RFC4474], section 9, for the SIP headers to hash to create digest-string.

3. Authentication using the Resource Owner Password Credentials flow

The resource owner password credentials flow is used by a Confidential UA with rich UI to authenticate to a third-party IdP entity and to directly obtain tokens to be able to register and get service from the SIP network.

3.1. Overview

The following figure provides a high level view of flow of messages for the OAuth Resource Owner Password Credentials flow:



3.2. Initial Registration

The UA first contacts the Authorization Server to authenticate the user and obtain tokens to be used to get access to the SIP network. The method used by the UA to obtain the tokens is out of scope for this document.

The UA starts the registration process with the SIP proxy by sending a REGISTER request (F1) with the access token it obtained previously.

The UA includes an Authorization header field with the Bearer scheme in the request to carry the access token obtained previously.

The proxy then validates the token, and MAY perform an introspection step to get more information about the token and its scope. The introspection step is out of scope for this document.

When the proxy is satisfied with the token, it then replies with the 200 OK to complete the registration process.

3.3. Subsequent Requests

All subsequent requests from the UA MUST include a valid access token. The UA MUST obtain a new access token before the access token expiry period to continue to get service from the system.

4. Authorization Header Syntax

This section describes the syntax of the authorization header with the Bearer scheme.

```
Authorization = "Authorization" HCOLON "Bearer" LWS
                "access_token" EQUAL access_token
access_token = quoted-string
```

5. Security Considerations

As this document uses the mechanism defined in the OAuth 2.0 [RFC6749], many of the security consideration in the OAuth 2.0 document apply to this document too.

The shared-key mechanism used with the Public UA allows a UA to re-register without the need to obtain a new access code. If this shared-key is leaked, an adversary will be able to register a UA and impersonate the attacked user.

To reduce the chances of the shared-key being leaked, the UA should not store the shared-key in a permanent storage, but keep it in memory only.

A server should limit the use of shared-key to clients that are able to provide an adequate level of protection for the shared-key. In some deployments, the server might decide not to support the use of shared-key at all.

6. IANA Considerations

6.1. Shared Key Feature-Capability Indicator

This document defines the feature capability sip.shared-key in the "SIP Feature-Capability Indicator Registration Tree" registry defined in [RFC6809].

Name: sip.shared-key

Description: This feature-capability indicator, when included in a Feature-Caps header field of a REGISTER response, indicates that the server supports the use of shared-key mechanism.

Reference: [this document]

7. Acknowledgments

The authors would like to thank the following for their review and feedback:

Andrew Allen, Martin Dolly, Keith Drage, Paul Kyzivat, Jon Peterson, Michael Procter, Roy Radhika, Matt Ryan, Ivo Sedlacek, Roman Shpount, Robert Sparks, Asveren Tolga, and Dale Worley.

Special thanks to Jon Peterson for a long discussion on the ideas and concepts around the use of OpenID/OAuth as an authentication and authorization mechanisms in a SIP network.

8. References

8.1. Normative References

[MITKB] "IdP (Identity Provider)", MIT Knowledge Base, http://kb.mit.edu/confluence/x/XoK2, March 2011.

[OPENID] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", February 2014.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, H., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", August 2006.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7662] Richer, J., "OAuth 2.0 Token Introspection", RFC 7662, October 2015.

8.2. Informative References

[RFC474bis] Peterson, J., Jennings, C., Rescorla, E., and C. Wendt, "Authenticated Identity Management in SIP", https://datatracker.ietf.org/doc/draft-ietf-stir-rfc4474bis/, February 2017.

Authors' Addresses

Rifaat Shekh-Yusef (editor)
Avaya
250 Sidney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5176
EMail: rifaat.ietf@gmail.com

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: christer.holmberg@ericsson.com

Victor Pascual
webrtchacks
Spain

EMail: victor.pascual.avila@gmail.com

SIP Core
Internet-Draft
Updates: 3261 (if approved)
Intended status: Standards Track
Expires: March 22, 2018

R. Shekh-Yusef
Avaya

September 18, 2017

The Session Initiation Protocol (SIP) Digest Authentication Scheme
draft-yusef-sipcore-digest-scheme-06

Abstract

This document updates the Digest Access Authentication scheme used by the Session Initiation Protocol (SIP) to add support for SHA2 digest algorithms to replace the MD5 algorithm.

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 March 22, 2018.

Copyright Notice

Table of Contents

1. Introduction

The SIP protocol [RFC3261] uses the same mechanism used by the HTTP protocol for authenticating users, which is a simple challenge-response authentication mechanism that allows a server to challenge a client request and allows a client to provide authentication information in response to that challenge.

The SIP protocol uses the Digest Authentication scheme that is used with the HTTP authentication mechanism, which by default uses MD5 as the default algorithm.

The HTTP Digest Access Authentication [RFC7616] document defines the Digest Authentication scheme and defines a few algorithms that could be used with the Digest Authentication scheme, and establishes a registry for these algorithms to allow for additional algorithms to be added in the future.

This document updates the Digest Access Authentication scheme used by SIP to add support for SHA2 digest algorithms to replace the MD5 algorithm.

1.1. Terminology

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

2. The SIP Digest Authentication Scheme

This section describes the modifications to the operation of the Digest mechanism as specified in [RFC3261] in order to support the SHA- 256 and SHA-512/256 algorithms as described in [RFC7616], and also to require support for the "qop" option."

2.1. Hash Algorithms

The Digest scheme has an ’algorithm’ parameter that specifies the algorithm to be used to compute the digest of the response. The IANA registry named "HTTP Digest Hash Algorithms" specifies the algorithms that correspond to ’algorithm’ values, and specifies a priority for each algorithm.

[RFC3261] specifies only one algorithm, MD5, which is used by default. This document extends [RFC3261] to allow use of any registered algorithm.

The priority of the algorithm defines its usage preference. UAs SHOULD prefer algorithms with higher priorities.

Note that [RFC7616] defines a -sess variant for each algorithm; the -sess variants are not used with SIP.

2.2. Representation of Digest Values

The size of the digest depends on the algorithm used. The bits in the digest are converted from the most significant to the least significant bit, four bits at a time to the ASCII representation as follows. Each four bits is represented by its familiar hexadecimal notation from the characters 0123456789abcdef, that is binary 0000 is represented by the character ’0’, 0001 by ’1’ and so on up to the representation of 1111 as ’f’. If the MD5 algorithm is used to calculate the digest, then the digest will be represented as 32

hexadecimal characters, SHA-256 and SHA-512/256 by 64 hexadecimal characters.

2.3. The Authenticate Response Header

When a UAS receives a request from a UAC, and an acceptable Authorization header is not sent, the UAS can challenge the originator to provide credentials by rejecting the request with a 401/407 status code with the WWW-Authenticate/Proxy-Authenticate header field. The UAS MAY include multiple WWW-Authenticate/Proxy-Authenticate headers to allow the UAS to utilize the best available algorithm supported by the client.

If the UAS challenges with multiple WWW-Authenticate/Proxy-Authenticate headers with the same realm, then each one of these headers MUST use a different digest algorithm. The UAS MUST add these headers to the response in the order that it would prefer to see them used, starting with the most preferred algorithm at the top, followed by the less preferred algorithms.

2.4. The Authorization Request Header

When the UAC receives a response with multiple headers with the same realm it SHOULD use the topmost header that it supports, unless a local policy dictates otherwise. The client MUST ignore any challenge it does not understand.

When the UAC receives a 401 response with multiple WWW-Authenticate headers with different realms it SHOULD retry and include an Authorization header containing credentials that match the topmost header of any one of the realms.

If the UAC cannot respond to any of the challenges in the response, then it should abandon attempts to send the request; e.g., if the UAC does not have credentials for any of the realms.

2.5. Forking

Section 22.3 of [RFC3261] discusses the operation of the proxy-to-user authentication, which describes the operation of the proxy when it forks a request. This section introduces some clarification to that operation.

If a request is forked, various proxy servers and/or UAs may wish to challenge the UAC. In this case, the forking proxy server is

responsible for aggregating these challenges into a single response. Each WWW-Authenticate and Proxy-Authenticate value received in responses to the forked request MUST be placed into the single response that is sent by the forking proxy to the UA.

When the forking proxy places multiple WWW-Authenticate and Proxy-Authenticate header fields from one received response into the single response it MUST maintain the order of these header fields. The ordering of the header field values from the various proxies is not significant.

2.6. HTTP Modifications

This section describes the modifications and clarifications required to apply the HTTP Digest authentication scheme to SIP. The SIP scheme usage is similar to that for HTTP. The changes specified here are mostly copied from section 22.4 of [RFC3261] with few changes.

SIP clients and servers MUST NOT accept or request Basic authentication.

The rules for Digest authentication follow those defined in HTTP, with "HTTP/1.1" replaced by "SIP/2.0" in addition to the following differences:

1. The URI included in the challenge has the following BNF:

   URI = Request-URI

2. The ’uri’ parameter of the Authorization header field MUST be enclosed in quotation marks.

3. The BNF for digest-uri-value is:

   digest-uri-value = Request-URI

4. The example procedure for choosing a nonce based on Etag does not work for SIP.

5. The text in [RFC7234] regarding cache operation does not apply to SIP.

6. [RFC7616] requires that a server check that the URI in the request line and the URI included in the Authorization header field point to the same resource. In a SIP context, these two URIs may refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the Request-URI in the

Authorization header field value corresponds to a user for whom the server is willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are not equivalent.

7. As a clarification to the calculation of the A2 value for message integrity assurance in the Digest authentication scheme, implementers should assume, when the entity-body is empty (that is, when SIP messages have no body) that the hash of the entity-body resolves to the hash of an empty string:

```
H(entity-body) = <algorithm>("")
```

For example, when the chosen algorithm is SHA-256, then:

```
H(entity-body) = SHA-256("") =
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
```

8. Servers MUST be able to properly handle "qop" parameter received in an authorization header field, and clients MUST be able to properly handle "qop" parameter received in WWW-Authenticate and Proxy-Authenticate header fields. Servers MUST always send a "qop" parameter in WWW-Authenticate and Proxy-Authenticate header field values, and clients MUST send the "qop" parameter in any resulting authorization header field.

The usage of the Authentication-Info header field continue to be allowed, since it provides integrity checks over the bodies and provides mutual authentication.

3. Augmented BNF for the SIP Protocol

This document updates the Augmented BNF for the SIP Protocol as follows.

It extends the request-digest as follows to allow for different digest sizes:

```
request-digest = LDQUOT *LHEX RDQUOT
```

The number of hex digits must be specified by the specification of the algorithm used.

It extends the algorithm parameter as follows to allow for SHA2 algorithms to be used:

```
algorithm = "algorithm" EQUAL ( "MD5" / "SHA-512-256" / "SHA-256"
/ token )
```

4. Security Considerations

This specification adds new secure algorithms to be used to with the Digest mechanism to authenticate users, but leaves the broken MD5 algorithm for backward compatibility.

This opens the system to the potential of a downgrade attack by man-in-the-middle. The most effective way of dealing with this type of attack is to remove the support for backward compatibility.

See section 5 of [RFC7616] for a detailed security discussion of the Digest scheme.

5. IANA Considerations

[RFC7616] defines an IANA registry named "Hash Algorithms for HTTP Digest Authentication" to simplify the introduction of new algorithms in the future. This document will use the algorithms defined in that registry.

6. Acknowledgments

The author would like to thank the following individuals for their careful reviews, comments, and suggestions: Paul Kyzivat, Olle Johansson, Dale Worley, Michael Procter, Inaki Baz Castillo, and Tolga Asveren.

7. Normative References

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, H., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC7234] Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.

[RFC7616] Shekh-Yusef, R., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, September 2015.

Author's Address

Rifaat Shekh-Yusef
Avaya
250 Sidney Street
Belleville, Ontario
Canada

Phone: +1-613-967-5176
EMail: rifaat.ietf@gmail.com