KITTEN
Internet-Draft
Updates: 2743, 2744 (if approved)
Intended status: Standards Track
Expires: October 1, 2017

N. Williams
Cryptonector

March 30, 2017

Channel Binding Signalling for the Generic Security Services Application Programming Interface

draft-ietf-kitten-channel-bound-flag-01

Abstract

Channel binding is a technique that allows applications to use a secure channel at a lower layer without having to use authentication at that lower layer. The concept of channel binding comes from the Generic Security Services Application Programming Interface (GSS-API). It turns out that the semantics commonly implemented are different that those specified in the base GSS-API RFC (RFC2743), and that that specification has a serious bug. This document addresses both, the inconsistency as-implemented and the specification bug.

This Internet-Draft proposes the addition of a "channel bound" return flag for the GSS_Init_sec_context() and GSS_Accept_sec_context() functions. Two behaviors are specified: a default, safe behavior reflecting existing implementation deployments, and a behavior that is only safe when the application specifically tells the GSS-API that it (the application) supports the new behavior. Additional API elements related to this are also added, including a new security context establishment API.

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 October 1, 2017.

Table of Contents

1. Introduction

The GSS-API [RFC2743] supports "channel binding" [RFC5056], a technique for detection of man-in-the-middle (MITM) attacks in secure channels at lower network layers. This facility is meant to be all-

or-nothing: either both the initiator and acceptor use it and it succeeds, or both must not use it. This has created a negotiation problem when retrofitting the use of channel binding into existing application protocols.

Many implementations of the Kerberos V5 GSS-API mechanism [RFC4121] cause the acceptor to succeed when the initiator used channel binding but the acceptor application did not. This has helped deployment of channel binding in existing applications: first fix all the initiators, then fix all the acceptors. But even this technique is insufficient when there are many clients to fix, such that fixing them all will take a long time.

This document proposes a new method for deployment of channel binding that allows the feature to be enabled on the acceptor side before fixing all initiators. If the GSS-API had always had a return flag by which to indicate channel binding state then we could have had a simpler method of deploying channel binding: applications check that return flag and act accordingly (e.g., fail when channel binding is required). We cannot safely introduce this behavior now without an indication of support by the application.

It is worth noting that at least one implementor of GSS-API mechanisms (but not of the GSS-API itself) has similar semantics in its API to those proposed herein. [XXX add references to the relevant SSPI docs? -Nico]

Additionally, there may be applications where it is important for initiators to know that acceptors did use channel binding, and even to know whether a mechanism is capable of indicating as much. We add a request flag and two mechanism attributes for such applications.

1.1. Error in RFC2743

The GSS-APIv2u1 [RFC2743] seems to indicate that mechanisms must ignore channel bindings when one party provided none. In practice some mechanisms ignore channel bindings when the acceptor provides none, but not when the initiator provides none. Note that it would be useless to allow security context establishment to succeed when the initiator does not provide channel bindings but the acceptor does, at least as long as there's no outward indication of whether channel binding was used! And indeed, the GSS-APIv2u1 does not provide any such indication. We correct this flaw in this document.

1.2. Design

After some discussion on the mailing list of various designs for signalling application support for the new flag we've settled on copying an aspect of the Java Bindings of the GSS-API [RFC5653], specifically the notion of creating an "empty" SECURITY CONTEXT handle that can then be passed to GSS_Init_sec_context() and GSS_Accept_sec_context() where they normally expect a NULL handle. This empty security context handle can then be used to set options relating to security context token establishment.

In [I-D.williams-williams-kitten-ctx-simple-async] we explore and extend this design to produce a more usable GSS-API (as well as support for asynchronous operation).

1.3. Alternative Design

An earlier design was based on an existing, non-standard extension for carrying security context establishment options in CREDENTIAL HANDLEs. A notion of CREDENTIAL HANDLE options might still be useful for options that are really specific to credentials rather than security context tokens, but for the time being we have no use for such a thing.

1.4. Future Directions

We're likely to introduce additional mutator functions of empty contexts, with mutators corresponding to many of the existing input arguments of GSS_Init_sec_context() and GSS_Accept_sec_context(), as well as a few additional security context inquiry functions. We're also likely to then introduce new variants of GSS_Init_sec_context() and GSS_Accept_sec_context() with all of those input and output parameters removed that could be set or retrieved with the other new functions. The only inputs that the new GSS_Init/ Accept_sec_context() must have are: a security context handle (never NULL), and an input context token, and the only outputs should be the status indicators and an output token. In fact, we may want to have just one new function called, perhaps, GSS_Step_sec_context(), with the role of initiator or acceptor set as a context option.

See [I-D.williams-williams-kitten-ctx-simple-async].

1.5. Conventions used in this document

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. Channel Binding State Extension

We propose a new return flag for GSS_Init_sec_context() and GSS_Accept_sec_context(), as well as a pair of functions for a) creating "empty" security context handles, b) setting req_flags and indicating which ret_flags the application understands. We also add new mechanism attributes describing mechanism capabilities.

C bindings of these extensions are provided along the lines of [RFC2744] and [RFC5587].

In the future we might move more of the many input (and output) arguments to GSS_Init_sec_context() and GSS_Accept_sec_context() into mutators on empty security context handles.

2.1. GSS_Create_sec_context()

Inputs:

- <none>

Outputs:

- major_status INTEGER
- minor_status INTEGER -- note: mostly useless, but we should keep it
- context SECURITY CONTEXT -- "empty" security context

Return major status codes:

- GSS_S_COMPLETE indicates success.
- GSS_S_UNAVAILABLE indicates that memory is not available, for example.
- GSS_S_FAILURE indicates a general failure.

This function creates an "empty" security context handle that can be passed to GSS_Init_sec_context() or GSS_Accept_sec_context() where they expect a NULL context.

2.1.1. C-Bindings

```
OM_uint32
gss_create_sec_context(OM_uint32 *minor_status,
                       gss_ctx_id_t *context);
```

2.2. GSS_Set_context_flags()

Inputs:

context CONTEXT HANDLE

req_flags FLAGS Requested flags. Applicable to acceptors and initiators.

ret_flags_understood FLAGS The set of return flags understood by the caller.

Outputs:

o major_status INTEGER

o minor_status INTEGER

Return major status codes:

o GSS_S_COMPLETE indicates success.

o GSS_S_FAILURE indicates a general failure.

This function tells the mechanism (when one is eventually chosen and invoked) that the application requests the given req_flags and undestands the given ret_flags. Initiators can override the req_flags in their GSS_Init_sec_context() call, but if no flags are requested there then the req_flags set on the empty context will be used.

NOTE: The abstract GSS-API [RFC2743] uses individual elements -one per-flag- instead of a "FLAGS" type. This is unwieldy, therefore we introduce an abstract type named "FLAGS" to act as a set of all the request/return flags defined for the abstract GSS-API.

2.2.1. C-Bindings

```
OM_uint32
gss_set_context_flags(OM_uint32 *minor_status,
                      gss_ctx_id_t context,
                      uint64_t req_flags,
                      uint64_t ret_flags);
```

2.3. Return Flag for Channel Binding State Signalling

Whenever both the initiator and the acceptor provide matching channel bindings to GSS_Init_sec_context() and GSS_Accept_sec_context(), respectively, then the mechanism SHALL indicate that the context is channel bound via an output flag, ret_channel_bound_flag, for the established context. Note that some mechanisms have no way for the acceptor to signal CB success to the initiator, in which case GSS_Init_sec_context() MUST NOT output the ret_channel_bound_flag.

2.3.1. C-Bindings

```
#define GSS_C_CHANNEL_BOUND_FLAG 2048 /* 0x00000800 */
```

2.4. New Mechanism Attributes

- We add a new mechanism attribute, GSS_C_MA_CBINDING_CONFIRM, to indicate that the initiator can and always does learn whether the acceptor application supplied channel bindings. Mechanisms advertising this attribute MUST always indicate acceptor channel bound state to the initiator.

- We add a new mechanism attribute, GSS_C_MA_CBINDING_MAY_CONFIRM, to indicate that the initiator may learn whether the acceptor application supplied channel bindings, but only when the acceptor implementation of the mechanism has been suitably updated. Mechanisms MUST advertise this attribute when the local initiator functionality for acceptor channel bound state indication exists but the acceptor may lack the same functionality (because, for example, the mechanism predates this specification).

OID assignments TBD.

2.5. Request Flag for Acceptor Confirmation of Channel Binding

We add a new request flag for GSS_Init_sec_context(), req_cb_confirmation_flag, to be used by initiators that insist on acceptors providing channel bindings. This flag is only of use to mechanism-negotiation pseudo-mechanisms (e.g., SPNEGO [RFC4178]): if set the pseudo-mechanism MUST NOT negotiate any mechanisms that lack the GSS_C_MA_CBINDING_CONFIRM or GSS_C_MA_CBINDING_MAY_CONFIRM mechanism attributes, and SHOULD NOT negotiate mechanisms that lack the GSS_C_MA_CBINDING_CONFIRM mechanism attribute (except if allowed by local configuration).

2.5.1. C-Bindings

Because GSS_C_CHANNEL_BOUND_FLAG is a return flag only, and this flag is a request flag only, and to save on precious flag bits, we use the same flag bit assignment for both flags:

```
#define GSS_C_CB_CONFIRM_FLAG 2048 /* 0x00000800 */
```

2.6. GSS_Delete_sec_context() Behavior When Applied to Empty Security Contexts

GSS_Delete_sec_context() MUST NOT output a context deletion token when applied to empty security contexts.

3. Modified Channel Binding Semantics

The channel binding semantics of the base GSS-API are modified as follows:

- Whenever both, the initiator and acceptor shall have provided input_channel_bindings to GSS_Init/Accept_sec_context() and the channel bindings do not match, then the mechanism MUST fail to establish a security context token. This is a restatement of an existing requirement in the base specification, restated for the reader's convenience.

- Whenever the acceptor application shall have a) provided channel bindings to GSS_Accept_sec_context(), and b) not indicated support for the ret_channel_bound_flag flag, then the mechanism MUST fail to establish a security context if the initiator did not provide channel bindings data. This requirement is critical for security purposes, to make applications predating this document secure, and this requirement reflects actual implementations as deployed.

- Whenever the initiator application shall have a) provided channel bindings to GSS_Init_sec_context(), and b) not indicated support for the ret_channel_bound_flag flag, then the mechanism SHOULD NOT fail to establish a security context just because the acceptor failed to provide channel bindings data. This recommendation is for interoperability purposes, and reflects actual implementations that have been deployed. It is possible that not all security mechanism protocols can implement this requirement easily.

- Whenever the application shall have a) provided channel bindings to GSS_Init_sec_context() or GSS_Accept_sec_context(), and b) indicated support for the ret_channel_bound_flag flag, then the mechanism MUST NOT fail to establish a security context just because the peer did not provide channel bindings data. The

mechanism MUST output the ret_channel_bound_flag if both peers provided the same input_channel_bindings to GSS_Init_sec_context() and GSS_Accept_sec_context. The mechanism MUST NOT output the ret_channel_bound_flag if either (or both) peer did not provide input_channel_bindings to GSS_Init/Accept_sec_context(). This requirement restores the original base GSS-API specified behavior, with the addition of the ret_channel_bound_flag flag.

4. Security Considerations

This document deals with security. There are no security considerations that should be documented separately in this section. To recap, this document fixes a significant flaw in the base GSS-API [RFC2743] specification that fortunately has not been implemented, and it adds a feature (that should have been in the base specification) for improved negotiation of use of channel binding [RFC5056].

5. IANA Considerations

Two GSS-API mechanism attributes are to be added to the "SMI Security for Mechanism gsscma Codes" registry established by RFC5587 [RFC5587]. See Section 2.4.

6. References

6.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <http://www.rfc-editor.org/info/rfc2743>.

[RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, DOI 10.17487/RFC2744, January 2000, <http://www.rfc-editor.org/info/rfc2744>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, DOI 10.17487/RFC5587, July 2009, <http://www.rfc-editor.org/info/rfc5587>.

6.2. Informative References

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, DOI 10.17487/RFC4121, July 2005, <http://www.rfc-editor.org/info/rfc4121>.

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, DOI 10.17487/RFC4178, October 2005, <http://www.rfc-editor.org/info/rfc4178>.

[RFC5653] Upadhyay, M. and S. Malkani, "Generic Security Service API Version 2: Java Bindings Update", RFC 5653, DOI 10.17487/RFC5653, August 2009, <http://www.rfc-editor.org/info/rfc5653>.

[I-D.williams-williams-kitten-ctx-simple-async] Williams, N., "Simplified and Asynchronous Security Context Interfaces for the Generic Security Services Application Programming Interface", draft-williams-williams-kitten-ctx-simple-async-00 (work in progress), February 2013.

Author's Address

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com