Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 13, 2012

Wilkinson
YFS

January 10, 2012

Integrating rxgk with AFS
draft-wilkinson-afs3-rxgk-afs-01

Abstract

This document describes how the new GSSAPI based rxgk security class for RX is integrated with the AFS application protocol. It describes a number of extensions to the basic rxgk protocol, clarifies a number of implementation issues, and provides values for the application specific elements of rxgk.

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 July 13, 2012.

Table of Contents

1. Introduction

rxgk [I-D.wilkinson-afs3-rxgk] is a new GSSAPI [RFC2743] based security layer for the RX [RX] remote procedure call system. The rxgk specification details how it may be used with a generic RX application, this document provides additional detail specific to integrating rxgk with the AFS-3 distributed file system.

1.1. The AFS-3 distributed file system

AFS-3 is a global distributed network file system. The system is split into a number of cells, with a cell being the administrative boundary. Typically an organisation will have one, or more cells, but a cell will not span organisations. Each cell contains a number of fileservers which contain collections of files ("volumes") which they make available to clients using the AFS-3 protocol. Clients access these files using a service known as the cache manager.

In order to determine which server a particular file is located upon, the cache manager looks up the location in the volume location database (vldb) by contacting the vlserver. Each cell has one or more vlservers, which are syncronised by an out-of-band mechanism.

1.2. rxgk and AFS

AFS-3 differs from the standard rxgk implementation in that it does not require GSSAPI negotiation with each server. Instead, a client negotiates with the vlserver, and receives a token which can then be used with any server in the cell. This requires that all servers have an identical cell wide pre-shared key for token encryption.

For more complex cell topologies, servers which do not share the cell-wide key are supported by means of an extended CombineTokens call. This call takes a server identifier, and will return a token encrypted with a key for a specific server. This extended call, AFSCombineTokens, also provides support for indicating whether a specific server is rxgk capable, allowing cells to securely migrate to rxgk from other security mechanisms.

We also define mechanisms for securing the callback channel which is created between fileserver and client.

1.3. Requirements Language

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 RFC 2119 [RFC2119].

2. Security Index

When used within the AFS protocol, rxgk has a securityIndex value of 4

3. Key negotiation

An AFS cell wishing to support rxgk MUST run an rxgk key negotiation service, as specified in [I-D.wilkinson-afs3-rxgk], on each of its vlservers. The service MUST listen on the same port as the vlserver.

The GSS identity afs-rxgk@_afs.<cellname> is the acceptor identity for this service. Where multiple vlservers exist for a single cell, all of these servers must have access to the key material for this identity, which MUST be identical across the cell. Clients MAY use the presence of this identity as an indicator of rxgk support for a particular cell. Clients which wish to support cells using other rx security objects MAY downgrade if this identity is not available.

Tokens returned from the GSSNegotiate call MUST only be used with database servers. Tokens for fileservers MUST be obtained by calling AFSCombineTokens before each server is contacted.

3.1. The AFSCombineTokens operation

AFS extends the existing CombineTokens operation to provide a more general token manipulation service. This operation takes a user token, an optional cache manager token, and a destination identifier, and returns a token specific to the specified destination.

```
AFSCombineTokens(IN RXGK_Data *token0<>,
                 IN RXGK_Data *token1<>,
                 IN afsUUID destination,
                 OUT RXGK_Data *new_token<>) = 3;
```

token0: An rxgk token obtained using the GSSNegotiate rpc

token1: Either, an rxgk token obtained using the GSSNegotiate rpc, or empty (0 length)

destination: The UUID of the server this token is intended for. Fileserver UUIDs may be obtained from the VLDB in the same call as returns their addresses.

new_token: A new rxgk token, or empty

The AFSCombineTokens call MUST only be performed over an rxgk protected channel, with a security level of 1 (integrity) or more. Servers MUST reject all attempts to perform this operation over channels that are not protected in this way.

Clients which are caching the results of RPCs on behalf of multiple users (such as a traditional AFS Cache Manager), SHOULD provide both the user's token (as token0) and a token generated from an identity that is private to the cache manager (as token1). This prevents a user from poisoning the cache for other users. Recommendations on keying cache managers are contained below

Clients which are working on behalf of a single user can provide an empty token1, but MUST use AFSCombineTokens to obtain a destination specific token for each fileserver they contact.

Clients using a printed token (see below) MUST provide that token as token0. token1 MUST be empty. Printed tokens cannot be combined with any other token, and servers MUST reject attempts to do so

If the returned token is 0 length, then the destination does not support rxgk, and the client MAY fall back to using a different authentication mechanism for that server. This is the only situation in which an rxgk capable client operating within an rxgk enabled cell may downgrade its choice of security layer.

Keys and tokens are combined in the same way as the CombineTokens call, documented in [I-D.wilkinson-afs3-rxgk].

4. Tokens

4.1. Container

rxgk tokens for AFS take the form of some key management data, followed by an encrypted data blob. The key management data (a version number, followed by an [RFC3961] encryption type) allows the recipient to identify which pre-shared key has been used to encrypt the token itself.

```
struct RXGK_TokenContainer {
  afs_int32 kvno;
  afs_int32 enctype;
  opaque    encrypted_token<>;
}
```

4.2. Token Encryption

Token contents are encrypted using a pre-shared key. rxgk supports the use of both a single cell-wide key and the use of per-server keys. The cell-wide key must be installed on all servers which are capable of accepting cell-wide tokens. Cell-wide keys should be for a selected RFC3961 encryption mechanism which is supported by all servers within the cell. Per-server keys should be for an encryption mechanism which is supported by both the destination server, and the negotiation service. The management of per-server keys is discussed in more detail below.

Key rollover is permitted by means of a key version number. When the key is changed, a different key version number MUST be selected. Servers SHOULD accept tokens using the old key until the lifetime of all existing tokens has elapsed.

Encryption is performed over the XDR encoded RXGK_Token structure, using the RFC3961 encrypt operation, with a key usage value of 1036 (RXGK_SERVER_ENC_TICKET)

4.3. Token Contents

The token itself contains the information expressed by the following XDR:

```
struct RXGK_Token {
  afs_int32 enctype;
  opaque    K0<>;
  afs_int32 level;
  afs_int64 starttime;
  afs_int32 lifetime;
  afs_int32 bytelife;
  rxgkTime expirationtime;
  struct PrAuthName identities<>;
}
```

enctype: The RFC3961 encryption type of the session key contained within this ticket

K0: The session key (see the rxgk specification for details of how this key is negotiated between client and negotiation service).

level: The security level that MUST be used for this connection

starttime: The time, expressed as a 100ns value, since the Unix epoch. Servers MUST reject attempts to start connections with tokens that are not yet valid.

lifetime: The maximum number of seconds that a key derived from K0 may be used for. This is an advisory limit. If 0, keys have no time based limit

bytelife: The maximum amount of data (expressed as log 2 byes) that may be transferred using a key derived from K0. This is an advisory limit. If 0, there is no data based limit on key usage

expirationtime: The time (expressed as an rxgkTime) beyond which this token may no longer be used. Servers MUST reject attempts to use connections secured with this token after this time has passed. A time of 0 indicates that this token never expires.

identities: A list of identities represented by this token. struct PrAuthName is the identity structure defined in [I-D.brashear-afs3-pts-extended-names]

5. Authenticator data

The appdata opaque within the RXGK_Authenticator contains the XDR encoded UUID of the client. The UUID is encoded using the afsUUID type.

6. Client tokens

In order to protect users of a multi-user cache manager from each other, it must be impossible for an individual user to determine the key used to protect operations which affect the cache. This requires that the cache manager have key material of its own which can be combined with that of the user. This functionality is provided by the AFSCombineTokens call specified earlier in this document. However, this call requires that a cache manager have access to a token for this purpose.

6.1. Keyed clients

Where a host already has key material for a GSSAPI mechanism supported by rxgk, that material may be used to key the client. The client simply calls the rxgk negotiation service using the relevant material, and obtains a token. The client should frequently renew this token, to avoid combined tokens having unnecessarily close

expiration times.

It is recommended that identities created specifically for use by a cache manager have the name afs3-callback@<hostname> where <hostname> is the fully qualified domain name of the cache manager.

6.2. Unkeyed clients

When a client has no key material, it is possible that an anonymous GSSAPI connection may succeed. Clients MAY attempt to negotiate such a connection by calling GSS_Init_Sec_Context() with the anon_req_flag [RFC2743] and the default credentials set.

7. Server to server communication

A number of portions of the AFS protocol require that servers communicate amongst themselves. To secure this with rxgk we require both a mechanism of generating tokens for these servers to use, and a definition of which identities are permitted for authorisation purposes.

7.1. Ticket printing

A server with access to the cell-wide pre-shared key may print its own tokens for server to server access. To do so, it should construct a token with suitable values. The list of identities in such a token MUST be empty. It can then encrypt this token using the pre-shared key, and use it in the same way as a normal rxgk token. The receiving server can identify it is a printed token by the empty identity list.

The session key within a printed token MUST use the same encryption type as the pre-shared key. When connecting to a fileserver, a client SHOULD use the combine tokens service as discussed above to ensure that they are using the correct key for the fileserver.

8. Declaring rxgk support for a fileserver

The AFSCombineTokens call has specific behaviour when a destination endpoint does not support rxgk. Implementing this behaviour requires that the vlserver be aware of whether a fileserver supports rxgk.

Fileservers currently register with the vlserver using the VL_RegisterAddrs RPC. Fileservers which support rxgk MUST call this RPC over a rxgk protected connection. The vlserver should then note the rx security layer used in registration, and infer rxgk support

from that. To prevent downgrade attacks, once a fileserver has registered as being rxgk capable, the vlserver MUST NOT remove that registration without administrator intervention.

Once a fileserver has been marked as supporting rxgk, VL_RegisterAddrs calls for that fileserver MUST only be accepted over an rxgk protected link.

9. Per server keys

The provision of servers with their own keys, rather than the cell wide master key, requires the ability to maintain a directory of these keys on the vlserver, so that the AFSCombineTokens RPC can encrypt the outgoing token with the correct key. The manner in which this directory is maintained is down to the implementor, who MAY decided to use a manual, or out of band key management system

Implementations supporting automatic key management through the AFS3 protocol MUST provide the following RPC

```
    struct RXGK_ServerKeyDataRequest {
        afs_int32 enctypes<>
        opaque nonce1<>
    };

    struct RXGK_ServerKeyDataResponse {
        afs_int32 enctype;
        afs_int32 kvno;
        opaque nonce2<>
    };

    VL_RegisterAddrsAndKey(
        IN afsUUID *uuidp,
        IN afs_int32 spare1,
        IN bulkaddrs *ipaddr,
        IN afs_int32 secIndex,
        IN opaque *keyDataRequest<>,
        OUT opaque *keyDataResponse<>) = XXX;
```

uuidp: As the existing VL_RegisterAddrs RPC

spare1: As the existing VL_RegisterAddrs RPC

ipaddr: As the existing VL_RegisterAddrs RPC

secIndex: The index of the security mechanism for which a key is being set. For rxgk, this value should be '4'

keyDataRequest: An opaque blob of data, specific to the security mechanism defined by secIndex. For rxgk it is, the xdr encoded representation of RXGK_ServerKeyDataRequest

keyDataResponse: An opaque blob of data, specific to the security mechanism defined by secIndex. For rxgk it is the xdr encoded representation of RXGK_ServerDataResponse

The client provides, in the RXGK_ServerKeyDataRequest structure, a list of the RFC3961 encryption types that it will accept as a server key. It also provides a nonce containing 20 random data bytes.

The server selects an encryption type shared by it and the client, and returns that, along with 20 bytes of random data that it has generated, in RXGK_ServerKeyDataResponse. If there is no common encryption type, then the server must fail the request.

The server key can then be derived by both client and server using

```
random-to-key(PRF+(K0, K, nonce1 || nonce2))
```

random-to-key is the function specified by the RFC3961 profile of the encryption type chosen by the server, and returned in enctype.

PRF+ is the function of that name specified by [RFC4402]

K0 is the master key of the current rxgk session, as originally determined by the GSSNegotiate call.

K is the key generation seed length as specified in enctype's RFC3961 profile

10. Securing the callback channel

AFS has traditionally had an unprotected callback channel. However, extended callbacks requires a mechanism for ensuring that callback breaks and, critically, data updates, are protected. This requires that there is a strong connection between the key material used initially to perform the RPC, and that which is used to protect any resulting callback. We achieve this using the cache manager token discussed earlier, which is required in order for a client to accept secure callbacks

A cache manager may set a key for secure callbacks by issuing the

following RPC (part of the RXAFS_ family)

```
RXAFS_SetCallbackKey(afs_int32 securityIndex,
                     opaque mech_data<>) = XXX;
```

securityIndex: The securityIndex of the mechanism for which this key is being set. In the rxgk case, this will be rxgk's security index, as defined earlier.

mech_data: This contains the security object specific data. In rxgk's case this is an XDR encoded RXGK_Token structure.

When used with rxgk, this RPC MUST be performed over an rxgk protected link established using solely the cache manager's token. This connection MUST have a security level of 2 (encrypted).

If a fileserver receives a AFS_SetCallbackKey protected with a different cache manager identity than the previous call from that client, it MUST break all secure callbacks held by that client using the old key before this RPC completes.

Only RPCs issued over an rxgk protected connection should receive rxgk protected callbacks

The fileserver MUST only send rxgk protected callbacks when one of the identities performing the RPC establishing that callback matches the identity associated with that clients callback channel.

11. IANA Considerations

This memo includes no request to IANA.

12. Security Considerations

12.1. Downgrade attacks

Using the presence of a GSSAPI key to determine a cell's ability to perform rxgk is vulnerable to a downgrade attack, as an attacker may forge error responses. Cells which no longer support rxkad SHOULD remove their afs@REALM and afs/cell@REALM Kerberos keys.

12.2. Per server keys

The mechanism for automatically registering per server keys is potentially vulnerable, as it trades a short lived key (the rxgk session key, which protects the key exchange) for a long life one

(the server key)

12.3. Combined key materials

As described earlier, combined tokens are used to prevent cache poisoning attacks on multi-user systems. In order for this protection to be effective, cache managers MUST NOT provide user access to keys produced through the combine tokens operation, unless those keys will not be used by the cache manger itself.

13. References

13.1. Informational References

[RX] Zeldovich, N., "RX protocol specification".

13.2. Normative References

[I-D.brashear-afs3-pts-extended-names]
Brashear, D., "Authentication Name Mapping extension for AFS-3 Protection Service", draft-brashear-afs3-pts-extended-names-09 (work in progress), March 2011.

[I-D.wilkinson-afs3-rxgk]
Wilkinson, S., "rxgk: GSSAPI based security class for RX", draft-wilkinson-afs3-rxgk-00 (work in progress), January 2010.

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

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 4402, February 2006.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

Appendix A. Acknowledgements

RXGK has been the work of many contributors over the years. A partial list is contained in the previous document. All errors and omissions are, however, mine.

Appendix B. Changes

B.1. Since 00

Add references to RX and XDR specifications

Add introductory material on AFS

Change expirationTime to be expressed using the rxgkTime type

Document how encryption types are chosen for printed tokens, and how they are used against fileservers

Expand security considerations section to cover combined tokens

Rename AFS_SetCallbackKey as RXAFS_SetCallbackKey

Author's Address

Simon Wilkinson
Your File System Inc

Email: simon@sxw.org.uk