Chapter 13
Authentication Multiplexor

Abstract

The Authentication Multiplexor routes authentication requests to the appropriate authentication service provider.

Introduction

Because Client32 is independent of any authentication service, it must be able to authenticate names using any authentication provider. This capability requires that there be a module that registers authentication providers and multiplexes authentication requests. That module is the Authentication Multiplexor (AuthMux).

Authentication handles allow callers to provide username and password information once, and then use that information later to authenticate new connections using the same information.

Below is a diagram showing AuthMux, two supported authentication service providers (Bindery authentication and NDS authentication) and the other Client32 NLMs.

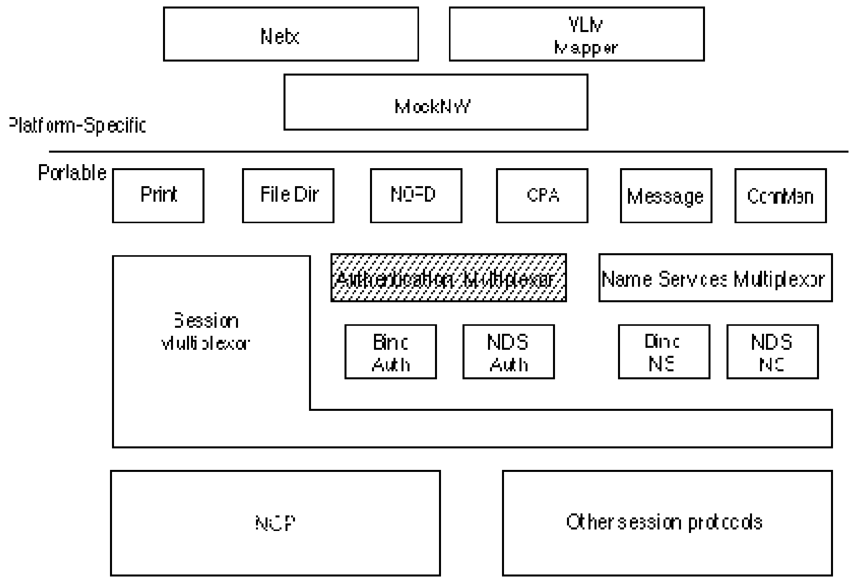

Figure 1. The Authentication Multiplexor routes authentication requests to any registered authentication provider, such as Bindery or NDS.

Design Description

AuthMux does three things: acts as a registry for authentication providers, multiplexes authentication requests to the appropriate authentication provider, and maintains a database of authentication handles.

Authentication Provider Registry

Authentication modules (for example, Bindery, NDS, PNW) must register their services with AuthMux so that a connection can be authenticated using a specific authentication method. (Bindery, NDS, and PNW all authenticate using different methods.) This scheme also allows a new authentication method (for example, retina scan) to be registered by an NLM for user verification.

When a provider registers with AuthMux, it passes a structure of pointers to functions (*AUTH_SVC_API_SET_TYPE*) that will be called by AuthMux.

The following is the complete set of authentication registry APIs that have been defined:

AUTHEnumerateAuthenticationSvc
AUTHRegisterAuthenticationSvc
AUTHUnregisterAuthenticationSvc

Authentication Multiplexing

There is an authentication provider for each supported name service, including NDS and Bindery. Each authentication provider must implement for its own name service the following set of routines:

AUTHAuthenticate
AUTHAuthenticateWithHandle
AUTHCloseAuthenticationHandle
AUTHCreateAuthenticationHandle
AUTHScanAuthenticateHandles
AUTHUnauthenticate

The structure for each routine is shown below.

```
typedef struct {

   UINT32   (*CreateAuthenticationHandle)
                    (UINT32      processGroupID,
                     UINT32      processID,
                     SPECT_DATA  *username,
                     SPECT_DATA  *password,
                     SPECT_DATA  *domainName,
                     VOID        *pAuthenSpecInfo,
                     AUTH_HANDLE *authenHandle );

   UINT32   (*AuthenticateWithHandle)
                    (AUTH_HANDLE  *authenHandle,
                     CONN_HANDLE  connHandle);

   UINT32   (*Authenticate)
                    (UINT32      processGroupID,
                     UINT32      processID,
                     CONN_HANDLE connHandle,
                     SPECT_DATA  *username,
                     SPECT_DATA  *password,
                     SPECT_DATA  *domainName,
                     VOID        *pAuthenSpecInfo);

   UINT32   (*Unauthenticate)
                    (CONN_HANDLE connHandle );

   UINT32   (*CloseAuthenticationHandle)
                    (AUTH_HANDLE *authenHandle );

   UINT32   (*GetAuthenInfo)
                    (AUTH_HANDLE *authenHandle,
                     SPECT_DATA  *username,
                     SPECT_DATA  *domainName,
                     VOID        *pAuthenSpecInfo);

} AUTH_SVC_API_SET_TYPE;
```

Management of Handle Database

The authentication handles returned by the various providers will be unique within the individual authentication scheme, but will not be unique across the whole system. That is, a handle returned by the Bindery authentication module may not be unique from one returned by NDS. AuthMux will keep a database to assign authentication handles that are unique system-wide.

NESL Events

AuthMux will produce events to inform interested NLMs of authenticated connections and unauthenticated connections.

EVENT_CONN_AUTHENTICATED

Syntax:	NESLProduceEvent(EVENT_CONN_AUTHENTICATED, conHandle);
Description:	AuthMux will produce this event after successfully authenticating the connection.

EVENT_CONN_UNAUTHENTICATED

Syntax:	NESLProduceEvent(EVENT_CONN_UNAUTHENTICATE D, connHandle);
Description:	AuthMux will produce this event after successfully unauthenticating the connection.