Microsoft Windows NT Internetworking Remote Access Support for Third-Party DLLs

Win32® Software Development Kit

For Microsoft® Windows®

Legal Information

Microsoft Windows NT Internetworking Remote Access Support for Third-Party Security DLLs

Introduction

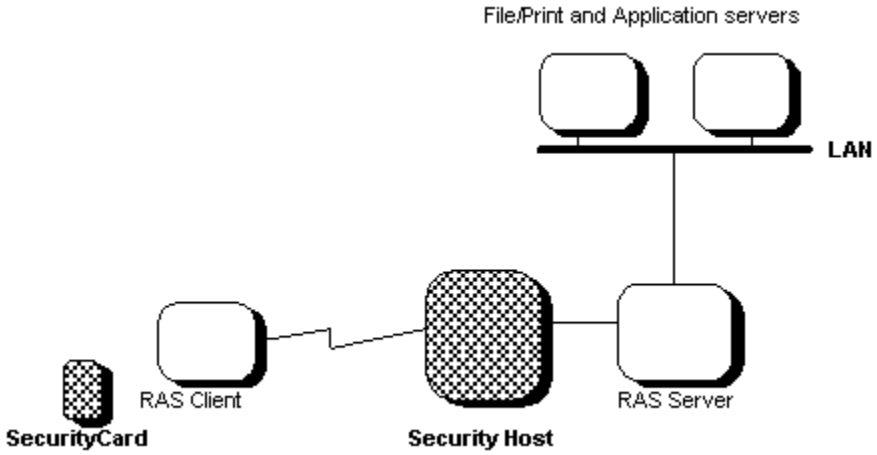
Large MIS shops want to enhance network access security by installing central host systems that use a combination of password cryptography and user security *card keys* to secure the network. For these customers, the fact that RAS has DES encryption and callback security is not enough: Callback security cannot be used for traveling executives, and DES encryption does not deter a hacker who knows a valid user's password.

A more secure mechanism would require an illegal user to have access to a valid user's *card key,* in addition to knowing the user's password. This is where third-party Security Hosts have an advantage over standard RAS security. These products typically include a Security Host on the back-end and a device like a card key, which will henceforth be called a SecurityCard.

This section briefly describes what was implemented for Microsoft® Windows NT® RAS version 3.1, and then describes enhancements for Microsoft® Windows NT® version 3.5*x.*

Windows NT version 3.1 - Windows for Workgroups version 3.11 RAS Solution

As shown in the following illustration, RAS provides a mechanism to allow the RAS Client to have a dialog with an intermediary Security Host system before going through RAS Server security. The RAS phone book enables this by allowing the user to go into terminal mode after a successful modem connection has been made, and before RAS authentication starts.



Third-party hardware security modules

The following table describes the typical connection sequence.

Step in Sequence	RAS Client Side	RAS Server Side (Security Host)
1	RAS user dials telephone number.	
2	Modems connect.	Modems connect but RAS Server is not yet notified of the connection.
3	RAS Phone book goes into terminal mode.	
4		Prompt for user name at RAS terminal. Send Challenge to RAS terminal. Prompt for response to Challenge.
5	User inputs Personal ID Number (PIN) to SecurityCard.	
6	SecurityCard becomes active	
7	User inputs Challenge to SecurityCard.	
8	SecurityCard scrambles Challenge using user-specific cryptographic	

	key.	
9	SecurityCard displays response on its output screen.	
10	User inputs response to terminal screen.	
11		If response is correct, notify RAS Server of connection. Else, disconnect.

Supported Hosts

Windows NT version 3.1 was tested with the following Security Hosts:

- Defender D1000 from Digital Pathways
- ACM400 from Security Dynamics
- Racal Gaurdata from Racal

Windows NT version 3.5*x* and Windows NT version 4.0 Support for Security DLLs

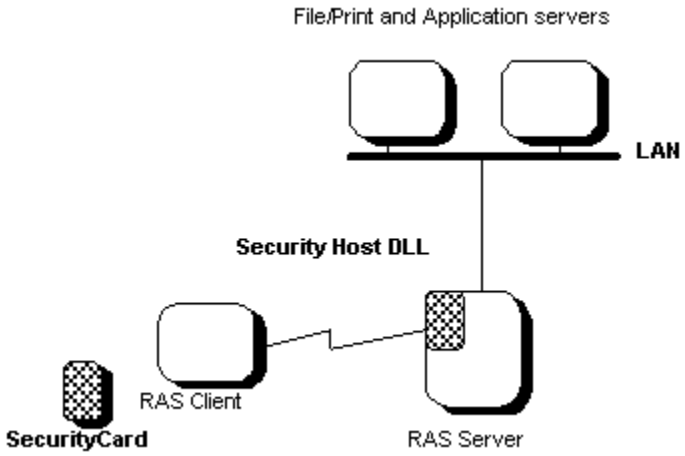
This section describes the RAS architecture that will enable software vendors to install their own Security DLLs on the RAS Server. This is a progression over and above what is currently available with Windows NT version 3.1 and Microsoft Windows for Workgroups version 3.11.

The scope of this section is limited to describing the framework that allows RAS to tie into diverse third-party security schemes. The goal of this section is not to provide an authentication scheme that allows RAS to dial into proprietary (non-PPP) remote access servers. The current RAS architecture for PPP authentication, framing, compression, encryption, and so on can and will be extended as needed to address proprietary remote access servers.

Architectural Requirements

The enhanced Security Host architecture will add support for the following software-pluggable modules:

- **SecurityCard**. A software module that validates the user by reading cryptographic keys from a floppy disk, hard disk, parallel port, retinal eye scanner, or other peripheral attached to the remote computers.
- **Security Host**. A software module that validates the user by reading security information from a database other than the standard Windows NT/RAS user account database. This host might work with hardware cryptographic keys such as what Windows NT version 3.1 and Windows for Workgroups version 3.11 support today through terminal mode. The key difference between this and current systems is that this system allows the RAS Server to be solely responsible for managing wide area network communications.

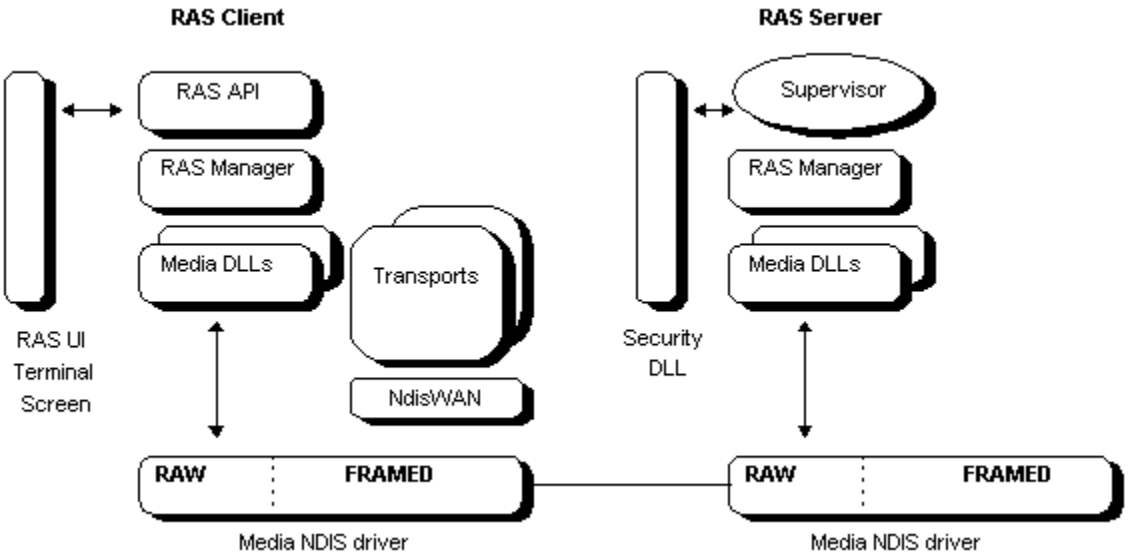


Third-party software security modules

The key requirements for allowing software-pluggable Security DLLs in RAS are:

- Simple Provider interface for Security DLLs. This interface should have a *black box* approach that isolates RAS from knowledge of various security schemes, isolates Security DLLs from the complexities of wide area networking, and is very easy to implement. Security DLLs will be responsible for writing logs directly to the Windows NT event log.
- RAW communication mode during RAS authentication, at which time Security DLLs are given a *pipe* through which to communicate. The Security DLLs are free to use whatever framing and encryption formats they need.

Architecture



Security DLL architecture

The server side sequence of the architecture is as follows:

1. The Security DLL is invoked immediately after a connection is established, but before any PPP or RAS authentication takes place. Only one Security DLL is active during the life of the RAS Server. RAS will not support the ability to cascade through multiple Security Hosts.
2. RAS API (or supervisor) queries the RAS connection Manager for a handle to pass to the Security DLL.
3. RAS API (or supervisor) passes the port handle to the Security DLL. The Security DLL may use this port handle to transmit and receive authentication information from the calling client.
4. The Security DLL validates the user and returns one of the following to RAS:
   - SUCCESS: The caller has been validated
   - FAIL: The caller has been denied access
   - ERROR: The Security Host DLL encountered an error and it could not authenticate the caller. Access will be denied.
5 RAS API (or supervisor) continues with the connection sequence, that is, moves on to RAS authentication

Note In this release of RAS, third-party Security DLLs will work only over asynchronous connections. In the future, other media such as ISDN will also be supported.

Windows NT Security Considerations

FRAMING mode authentication *always* happens after CONNECT COMPLETE. This ensures that the remote access servers security is never bypassed, regardless of the installed Security Host DLL. This is required to protect the network from a Trojan horse attack (for example, a bogus Security DLL pretends to be the end-all security authority and allows everyone into the network).

Administrators who want to use a separate user account database than that provided by Windows NT may do so by giving the Windows NT Guest account RAS permissions, and letting all RAS users know the password to the Guest account. This minimizes the administration overhead required on the Windows NT system, though it does not reduce this overhead to zero.

Installation Considerations

Users will install third-party security software separately from the RAS installation. The installation programs for these software modules will register themselves to RAS by means of the Windows NT Registry. The configured Security Host DLL will be loaded by the RAS service as soon as it starts. When a new call is received, the RAS Manager will invoke the registered Security DLL after it has done preliminary validation against the local or domain account database.

Registry Definitions

Security Host DLLs must register with RAS by writing the following information into the Windows NT Registry:

HKEY_LOCAL_MACHINE\Software\Microsoft\RAS\SecurityHost

DLLPath = REG_SZ <full path to DLL down to example.dll file name>

DLLPath Registry value. Lists the complete path of the Security DLL on the system.

The DLL path does not need to include the directory information if it resides in a directory listed in the system PATH.

Security vendors are responsible for providing setup programs that install their DLLs and register with RAS in the Windows NT Registry. Vendors should provide *remove/uninstall* functionality in their setup programs, and must deregister with RAS by deleting their entries in the Registry when the user removes the Security DLL from the system. Removing the DLL from the system without cleaning up the Registry will prevent the RAS service from starting.

Note Only one Security DLL may be installed in the system. In the version of RAS in Windows NT version 3.5x, cascading authentication through multiple third-party DLLs is not supported.

Programming Considerations

All the functions and data structures used by the Security Host DLL to call into the system during caller authentication are defined in the RASSHOST.H header file, which must be obtained from Microsoft. An ISV developing a Security Host add-on for RAS must export the following entry points by name in its DLL:

- **RasSecurityDialogBegin( )**
- **RasSecurityDialogEnd( )**

The RAS Manager will call the **LoadLibrary( )** and **GetProcAddress( )** functions on the third-party DLL expecting these entry points. If they are not available, the DLL will not be loaded.

Starting the Authentication

When a call is received, the RAS Manager will call into the Security Host through **RasSecurityDialogBegin( )** and pass a list of parameters for the current session. The Security Host should not block the call; instead, it should copy the passed parameters and dispatch a worker thread to do the actual authentication and communication with the client responder. If the call to the Security Host entry point is blocked, RAS will also block and no other calls will be accepted. The RAS supervisor guarantees the validity of the pointers passed in for as long as the Security Host is active, up until it calls the callback function indicated by RAS to terminate the authentication transaction.

Transmitting and Receiving Data

During the authentication of a caller, the third-party Security Host is given a handle to the communications port through which it can send and receive information from the client-side security add-on. This handle is not a regular handle that may be used with the Win32 I/O functions; instead, the Security Host will use two functions defined by the RAS Manager for sending and receiving data over the asynchronous line. These functions are:

- **RasSecurityDialogSend( )**
- **RasSecurityDialogReceive( )**

RasSecurityDialogSend( ) and **RasSecurityDialogReceive( )** are not available for static import from any library. They must be obtained using **LoadLibrary( )** on the RASMAN.DLL and then using **GetProcAddress( )** to obtain function pointers to them. The prototype for these function pointers are as follow:

```
typedef DWORD (WINAPI * RASSECURITYDIALOGSENDPROC)
(HPORT, PBYTE, WORD);

typedef DWORD (WINAPI * RASSECURITYDIALOGRECEIVEPROC)
(HPORT, PBYTE, PWORD, DWORD, HANDLE);
```

RasSecurityDialogSend( )

RasSecurityDialogSend( ) is used to send a block of data from the Security Host on the server to the client responder. The function takes as a parameter the handle to the port passed in during the **RasSecurityDialogBegin( )** function. The pointer to the buffer that will be sent must be the buffer that was passed in during **RasSecurityDialogBegin( )** also. When a host needs to transmit any data, it must first copy it to this buffer passed in by the RAS Manager and then call the send function to actually transmit it. The buffer has a limit (about 1.5 Kb) specified by the parameter in the entry point to the host DLL.

This function is asynchronous and returns the status error PENDING (defined in the RASERROR.H file), indicating the request is in progress.

RasSecurityDialogReceive( )

RasSecurityDialogReceive( ) is used to receive a block of data from the client responder. The function takes as a parameter the handle to the port passed in during the **RasSecurityDialogBegin( )** function. The received data is returned in the buffer passed, which must be the receive buffer indicated during **RasSecurityDialogBegin( )** also. When the host receives data, it must copy it to this buffer passed into its own data structures or buffer. The buffer has a limit (about 1.5 Kb) specified by the parameter in the entry point to the host DLL.

This function is asynchronous and returns the status error PENDING (defined in the RASERROR.H file), indicating the request is in progress.

When the Security Host is expecting data from the client responder, it may block until the data is received. The **RasSecurityDialogReceive( )** function facilitates this by allowing the caller to specify an event that will be set when the data is received through the port. After calling the transmitting function the caller may wait for this event to be set. The wait way be indefinite or it may have a time-out period, after which the host may assume the client did not respond. Another facility yet available from the **RasSecurityDialogReceive( )** function is that a client may specify a period of time (in seconds) that the system will wait for incoming data before setting the event passed in. When this parameter is set to a non-zero value, the handle to the event must be valid. When this period elapses, the RAS Manager will set the event, thus unblocking the caller.

Returning the Authentication Result

When the host determines the authentication of a calling user, it returns the result to RAS by calling the function whose pointer was passed in during **RasSecurityDialogSend( )**.

To return this information, the client must fill a structure defined as follows:

```
typedef struct _SECURITY_MESSAGE
{
    DWORD dwMsgId;
    HPORT hPort;
    DWORD dwError;        // Should be non-zero only if error
                          // occurred during the security dialog
                          // Should contain errors from WINERROR.H
                          // or RASERROR.H
    CHAR  UserName[UNLEN+1]; // Should always contain username if
                             // dwMsgId is SUCCESS/FAILURE
    CHAR  Domain[DNLEN+1];   // Should always contain domain if
                             // dwMsgId is SUCCESS/FAILURE
} SECURITY_MESSAGE, *PSECURITY_MESSAGE;
```

In the **dwMsgID** member, the security returns one of the following codes:

- SECURITYMSG_SUCCESS: The user has been granted access.
- SECURITYMSG_FAILURE: The user has been denied access.
- SECURITYMSG_ERROR: There was an error; an access was not determined. Default to no access.

The host is responsible for obtaining at least the name of the user for the *UserName* parameter. This parameter must be equal to the account name in the local or domain account database for the user calling in with RAS access. If the security returns denied access or an error, the RAS Manager will make an appropriate entry in the SYSTEM event log.

When the Security Host DLL calls into the RAS Manager, the system will initiate a cleanup sequence to free the buffers and the state in which the system is prior to switching to PPP. Part of this sequence is to call into the host DLL **RasSecurityDialogEnd( )** entry point. During this call the host should return a non-zero error to indicate that it has finished processing. If the Security DLL needs to stall the termination, it would return NO_ERROR and later on call into the RAS Manager in the **RasSecurityDialogComplete( )** function.

Aborting the Current Session

If for some reason the RAS Manager needs to terminate the current session, it will call into the Security Host DLL in its **RasSecurityDialogEnd( )** exported entry point. The Security Host is then responsible for terminating the worker thread associated with the port handle indicated by the passed parameter. If the **RasSecurityDialogEnd( )** function returns a non-zero error, the Security Host is not responsible for calling the termination callback function. However, if the function returns zero as the result, the RAS Manager will expect to be invoked in the termination callback to properly clean up the session.

Sample Code: RASHOST.DLL

RASHOST.DLL is a sample skeletal implementation of a Security Host DLL. This sample demonstrates the guidelines outlined in these topics. The sample does not actually authenticate a user, but it demonstrates how the entry points are used and how to deny or accept a calling user.