RSA SecurID Software Token 3.0
for Windows® Workstations
Developer's Guide

Introduction

The RSA SecurID Software Token 3.0 Application Programming Interface (API) enables third-party vendors to develop an interface to RSA SecurID Software Token 3.0. Any program that can use Windows DLLs can start the Software Token application and retrieve PASSCODEs directly. Such programs can be written, for example, in Microsoft Visual C++ or Microsoft Visual Basic.

Because copying an RSA SecurID software token from one computer to another computer can breach security, the software token is locked to the computer on which it is installed if the copy protection option was selected when the token was issued. If the token is copy protected, only an RSA SecurID Software Token administrator can uninstall a software token and install it on another computer.

Supported Environments

Windows 98, Windows NT, Windows 2000, and Windows XP

RSA SecurID Software Token 3.0 API Contents

The RSA SecurID Software Token 3.0 API package contains:

- Developer's Guide – this document (**stauto32.pdf**)
- Header file (**stauto32.h**)
- 32-bit DLL (**stauto32.dll**)
- 32-bit LIB file (**stauto32.lib**)
- A sample console application written using Microsoft Visual C++

Installing the RSA SecurID Software Token 3.0 API

The RSA SecurID Software Token 3.0 API DLL (**stauto32.dll**) is automatically installed with RSA SecurID Software Token 3.0 software.

Upgrading Previous Versions of the API to 3.0

The RSA SecurID Software Token 3.0 installation software automatically upgrades earlier versions of the API DLL.

Note: The RSA SecurID Software Token 3.0 API has not changed since 2.0.

RSA SecurID Software Token 3.0 API Features

The RSA SecurID Software Token 3.0 API supports multiple instances of the token service and the ability to handle multiple token devices.

- To include multiple instances of the token service, a handle has been implemented to identify each instance. Your program must call **OpenTokenService** and store the handle that is returned. Your program must use this handle in all future calls to the API and call **CloseTokenService** to close the instance of the API.
- The API supports multiple tokens. The **EnumToken** function enumerates all the available tokens provided by the token service. After the tokens have been enumerated, the program can either use the default token already selected or call **SelectToken** to change the current token.

Using the API

The following table shows how the API is typically used.

Call	Function
OpenTokenService	Open an instance of the token service and get a handle to it.
EnumToken	List the available tokens and allow the user to select the token to use.
SelectToken	After the user has selected a token, select the token.
GetPasscode	Ask the user for a PIN and retrieve the PASSCODEs. • PASSCODE mode — Send PASSCODE to authenticate. • Next PASSCODE mode — Send PASSCODE and Next PASSCODE to authenticate. • New PIN mode — Send PASSCODE, ask user for new PIN, call GetPasscode with PIN, and then send Next PASSCODE.
CloseTokenService	Close the token service.

RSA SecurID Software Token 3.0 API Functions

This section describes the following functions:

- **OpenTokenService**
- **EnumToken**
- **SelectToken**
- **SetTokenTime**
- **GetPasscode**
- **GetTokenError**
- **GetDLLInfo**
- **CloseTokenService**

OpenTokenService

int WINAPI OpenTokenService(LPLONG *pServiceHandle*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

pServiceHandle

[out] Pointer to a LONG that will receive the handle of the token service started if the function succeeds. This handle must be used in all future calls to the token service. If another program or process is using the DLL, it will be assigned its own handle that will allow parallel use of the token service.

Remarks

This function creates an instance to the API and returns a service handle, initializes the DLL, and starts the RSA SecurID Software Token service.

Example

```
LONG lTokenServiceHandle;

OpenTokenService(&lTokenServiceHandle);
```

EnumToken

int WINAPI EnumToken(LONG *lServiceHandle*, LPLONG *plNumberOfTokens*, LPLONG *plDefaultToken*, LPTOKENBASICINFO *pBuffer*, LPDWORD *pdwBufferSize*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

plNumber OfTokens

[out] Pointer to a LONG that will receive the number of tokens found.

plDefaultToken

[out] Pointer to a LONG that will receive the zero-based index of the default token.

pBuffer

[out] Pointer to a buffer that will receive the enumerated tokens. This buffer must be a pointer to an array of TOKENBASICINFO structures.

pdwBufferSize

[in] Pointer to a DWORD that contains the number of bytes allocated for *pBuffer*. Example: number of elements in array * sizeof(TOKENBASICINFO).

Remarks

This function enumerates all tokens found in the open token service. It populates an array of TOKENBASICINFO structures, where TOKENBASICINFO is defined as:

```
typedef struct tagTOKENBASICINFO
{
     DWORD dwSize;
     char serialnumber[24];
     char username[24];
     char deviceID[24];
     char descriptor[48];
} TOKENBASICINFO, FAR* LPTOKENBASICINFO;
```

If a token is password protected, the descriptor field for the token will be set to "PASSWORD." Applications must check the descriptor field of a token before it is selected and if it is password protected must query the user and pass the password to the **SelectToken** function. The deviceID field is not currently used.

Example

```
extern LONG lTokenServiceHandle;
LONG lTokens, lDefaultToken;
LPTOKENBASICINFO lpTokens = new TOKENBASICINFO[100];
DWORD dwBufferSize = 100 * sizeof(TOKENBASICINFO);

EnumToken(lTokenServiceHandle, &lTokens, &lDefaultToken, lpTokens,
&dwBufferSize);
```

SelectToken

int WINAPI SelectToken(LONG *lServiceHandle*, LPCSTR *pSerialNumber*, LPCSTR *pDeviceID*, LPCSTR *pPassword*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

pSerialNumber

[in] Pointer to a constant null-terminated string that contains the serial number of the token that will become the selected (default) token.

pDeviceID

[in] Not used. Applications may pass NULL for this parameter.

pPassword

[in] Pointer to a constant null-terminated string that contains the password of the token being selected. Required if the token is password protected (descriptor field = "PASSWORD"), otherwise applications may pass NULL.

Remarks

This function changes the default token to the token whose serial number matches the serial number passed in. When functions that use a token are called (for example, **GetPasscode**), the default token is used.

Example

```
extern LONG lTokenServiceHandle;
char chSerialNumber[24];
char chPassword[32];

strcpy(chSerialNumber, "26406048");
strcpy(chPassword, "My Secret Phrase");
SelectToken(lTokenServiceHandle, chSerialNumber, NULL, chPassword);
```

SetTokenTime

int WINAPI SetTokenTime(LONG *lServiceHandle,* LONG *lTime*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

lTime

[in] Current time expressed as the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Coordinated Universal Time.

Remarks

This function is used to pass the Software Token an accurate server time. Without this function, the current PC time (using the time zone setting plus the local time) is used to generate PASSCODEs. If the PC time is incorrect, the wrong PASSCODE is generated. With this function, an accurate server time can be passed down to the token, guaranteeing that the token is time synchronized with the server. This function eliminates all time-related authentication failures. The value of the time passed to the token equals the return value of the Microsoft time(0) function. See your Microsoft MSDEV documentation for the exact definition of this value.

Example

```
extern LONG lTokenServiceHandle;
time_t lTime;

time(&lTime);
SetTokenTime(lTokenServiceHandle, lTime);
```

GetPasscode

int WINAPI GetPasscode(LONG *lServiceHandle*, LPCSTR *pPin*, LPSTR *pPasscode*, LPSTR *pNextPasscode*, LPSTR *pTokencode*, LPSTR *pNextTokencode*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

pPin

[in] Pointer to a constant null-terminated string that contains the PIN to be used for generating PASSCODEs.

pPasscode

[out] Pointer to a string that will receive the current PASSCODE of the default token.

pNextPasscode

[out] Pointer to a string that will receive the next PASSCODE of the default token.

pTokencode

[out] Pointer to a string that will receive the current tokencode of the default token.

pNextTokencode

[out] Pointer to a string that will receive the next tokencode of the default token.

Remarks

This function is used retrieve the current and next PASSCODEs and tokencodes for the selected (default) token.

Example

```
extern LONG lTokenServiceHandle;
char chPIN[9];
char chPASSCODE[12];
char chNextPASSCODE[12];
char chTokencode[12];
char chNextTokencode[12];

strcpy(chPIN, "12345678");
GetPasscode(lTokenServiceHandle, chPIN, chPASSCODE, chNextPASSCODE,
chTokencode, chNextTokencode);
```

GetTokenError

int WINAPI GetTokenError(LONG *lServiceHandle,* LPTOKENERRORINFO *pErrorInfo*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

pErrorInfo

[out] Pointer to a TOKENERRORINFO structure that will receive information about the last problem that occurred in the token service.

Remarks

If any API function fails, your application can call this function to determine what has gone wrong. The TOKENERRORINFO structure is defined as:

```
typedef struct tagTOKENERRORINFO
{
     int error;
     char error_string[24];
     char detailed_error_string[64];
} TOKENERRORINFO, FAR* LPTOKENERRORINFO;
```

Example

```
extern LONG lTokenServiceHandle;
TOKENERRORINFO TokenError;

GetTokenError(lTokenServiceHandle, &TokenError);
```

GetDLLInfo

int WINAPI GetDLLInfo(LPDLLINFO *pDllInfo*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

pDllInfo

[out] Pointer to a DLLINFO structure that will receive information about the TokenAPI DLL.

Remarks

This function returns information about the TokenAPI DLL. The DLLINFO structure is defined as:

```
typedef struct tagDLLINFO
{
     long dll_version_num;
     long PRN_source;
} DLLINFO, FAR * LPDLLINFO;
```

Example

```
DLLINFO DllInfo;

GetDLLInfo(&DllInfo);
```

CloseTokenService

int WINAPI CloseTokenService(LONG *lServiceHandle*);

Return Value

If the function succeeds, the return value is greater than zero.

If the function fails, the return value is zero.

Parameters

lServiceHandle

[in] Handle that identifies the instance of the token service.

Remarks

This function closes the token service.

Example

```
extern LONG lTokenServiceHandle;
CloseTokenService(lTokenServiceHandle);
```

Sample Application

The following is a sample console application written in Visual C++ that demonstrates the use of the Software Token API. The source for this sample is provided with the Software Token API.

```
// TokenAPISample.cpp : Defines the entry point for the console application.
//
#include <stdio.h>
#include <windows.h>
#include "stauto32.h"
int main(int argc, char* argv[])
{
    printf("TokenAPI Sample Program...\n");
    LONG lTokenServiceHandle;
    LONG lTokens, lDefaultToken;
    LPTOKENBASICINFO lpTokens = new TOKENBASICINFO[100];
    DWORD dwBufferSize = 100 * sizeof(TOKENBASICINFO);
    char chPassword[32];
    char chPIN[9];
    char chPASSCODE[12];
    char chNextPASSCODE[12];
    char chTokencode[12];
    char chNextTokencode[12];

    // Open the Token Service and get a handle to it
    if(OpenTokenService(&lTokenServiceHandle) > 0)
    {
        printf("Token Service started, Handle=%ld\n", lTokenServiceHandle);
        // List tokens
        if(EnumToken(lTokenServiceHandle, &lTokens, &lDefaultToken, lpTokens, &dwBufferSize) > 0)
        {
            printf("%ld Software tokens found...\n", lTokens);
            LPTOKENBASICINFO lpToken = lpTokens;
            for(int i = 0; i < lTokens; i++)
            {
                printf("%d: Serial Number=%s\t User Name=%s\t Descriptor=%s\n",
                              i, lpToken->serialnumber, lpToken->username, lpToken->descriptor);
                lpToken++;
            }

            // Select the first token, if there is one
            if(lTokens > 0)
            {
                // Initialize password to NULL
                chPassword[0] = '\0';

                // If the token is password protected, prompt for password
                if(strncmp(lpTokens[0].descriptor, "PASSWORD", 8) == 0)
                {
                     printf("Enter password for token serial number %s:", lpTokens[0].serialnumber);
                     gets(chPassword);
                }

                // Select the token
                if(SelectToken(lTokenServiceHandle, lpTokens[0].serialnumber, NULL, chPassword) > 0)
                {
                     // Get PASSCODEs and tokencodes using the current PC time,
                     // with a PIN of "12345678"
```

```
                        strcpy(chPIN, "12345678");
                        GetPasscode(lTokenServiceHandle, chPIN,
                                    chPASSCODE, chNextPASSCODE, chTokencode, chNextTokencode);

                        printf("PASSCODE:\t%s\n",              chPASSCODE);
                        printf("Next PASSCODE:\t%s\n",   chNextPASSCODE);
                        printf("Tokencode:\t%s\n",             chTokencode);
                        printf("Next Tokencode:\t%s\n",  chNextTokencode);
                    }
                    else
                       {
                        printf("Failed to select token\n");
                     }
                }
            }

            // Close the Token Service
            CloseTokenService(lTokenServiceHandle);
            printf("Token Service stopped");
        }

        // Cleanup
        delete lpTokens;

        return 0;
    }
```

Sample Output:

```
TokenAPI Sample Program...
Token Service started, Handle=3151760
7 Software tokens found...
0: Serial Number=000050696808    User Name=Thomas Stefanick      Descriptor=PASSWORD
1: Serial Number=000050696809    User Name=Norik Kocharyan       Descriptor=
2: Serial Number=000050696810    User Name=Scott Tower           Descriptor=PASSWORD
3: Serial Number=000050696811    User Name=Kathe Rhoades         Descriptor=PASSWORD
4: Serial Number=000050696812    User Name=Juanella Wilson       Descriptor=
5: Serial Number=000050696813    User Name=Jason Cheng           Descriptor=
6: Serial Number=000050696814    User Name=Clarita Gonzales      Descriptor=
Enter password for token serial number 000050696808:My Secret Code
PASSCODE:       86893148
Next PASSCODE:  04034276
Tokencode:      74558570
Next Tokencode: 92799608
Token Service stopped
```