

Toughen Web application security

Build a multiphased authentication system with WebSEAL

Skill Level: Intermediate

[Christopher Hockings \(hockings@au1.ibm.com\)](mailto:hockings@au1.ibm.com)
Advanced Customer Engineering Team Member
IBM

01 May 2003

Carelessly chosen passwords have made many password-protected systems vulnerable to outside attack. This tutorial shows you how you can use Tivoli Access Manager WebSEAL to build a multiphased authentication system that locks Web applications down more tightly. The tutorial includes sample C code that you can use as a basis for your own applications.

Section 1. Introduction

What is this tutorial about?

This tutorial explains how to implement multiphased authentication methods using Tivoli Access Manager (TAM) WebSEAL. It provides an overview of multiphased authentication systems support within TAM WebSEAL, and presents a coded example for extending the capabilities to include other multiphased authentication systems.

You'll see the implementation of a cross-domain authentication service (CDAS) within WebSEAL, and follow a practical example using the mobile phone Short Message Service (SMS). This example uses the `token-cdas` interface provided within WebSEAL to simulate the multiphased authentication process.

Who should take this tutorial?

Before you start this tutorial, you should be familiar with the following:

- **Tivoli Access Manager installation and configuration:** You should have a solid understanding of the Tivoli Access Manager operating environment. This includes previous installation and configuration experience with the product. You should also be familiar with the CDAS interface, as this tutorial uses this interface to present an advanced authentication topic.
- **Solid C development:** You should be able to read and understand C code so that you can adapt the sample code to your operating environment. This tutorial presents code that shows a real example of using the token CDAS interface.
- **General Web experience:** Access to a SMS gateway is helpful. This tutorial uses a Web form to post a message to an SMS gateway.

Software requirements

To complete the examples shown in this tutorial, you will need the following installed on your system:

- **Windows development environment:** To compile the example CDAS code provided. You'll need to install Microsoft Visual Studio C++ 6.0 and the Microsoft WinHTTP 5.0 development package; the latter is for testing the implementation with an HTTP-enabled SMS gateway.
- **Access Manager WebADK package and its prerequisites:** This includes IBM Tivoli Access Manager Base V4.1 and IBM Tivoli Access Manager Web Security ADK V4.1
- **WebSEAL test environment:** To unit-test the implementation. You'll need the following components for this environment:
 - IBM Tivoli Access Manager Base V4.1: This includes a configured runtime and policy server. In this tutorial, we'll use the IBM Directory; however, any supported directory could be used in its place.
 - IBM Tivoli Access Manager WebSEAL V4.1: This includes a configured WebSEAL instance.

There's a link from which you can download this Tivoli software in [Resources](#). The following figure illustrates the test environment.

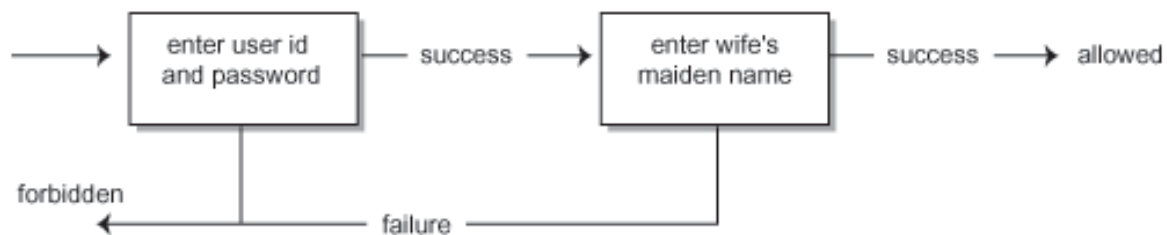
- **SMS gateway:** This component is not mandatory. However, the tutorial

code communicates with a gateway via HTTP to send the one-time password to the end user's mobile phone via SMS, so having such a gateway would be helpful.

Section 2. Multiphased authentication

What is multiphased authentication?

Multiphased authentication allows for chained authentication schemes to be executed in a sequence before an end user can proceed. The end user is expected to satisfy the authentication requirements of all of the schemes. The diagram below shows a simple multiphased authentication scheme -- which can be referred to as *two-factor authentication* because it includes only two phases -- that expects an end user to supply his user ID and password, followed by his wife's maiden name.



The end user must pass all of the authentication requests before proceeding to the success outcome.

Why use multiphased authentication?

Single-factor authentication methods such as passwords are no longer considered secure in the Internet world, as users like to use such easy-to-remember (and hence easy-to-guess) passwords as birthdays, children or family member names, pet names, etc. Unidentified hackers have deployed password-gathering programs over the Internet that have successfully collected millions of passwords. There is a rapid growth in demand for strong authentication on highly critical Web-based applications. Under strong authentication, the user must present authentication identifiers (or credentials) based on two or more factors. Traditionally, this requires end users to supply:

- Something that they *know*, such as a password or a PIN number.

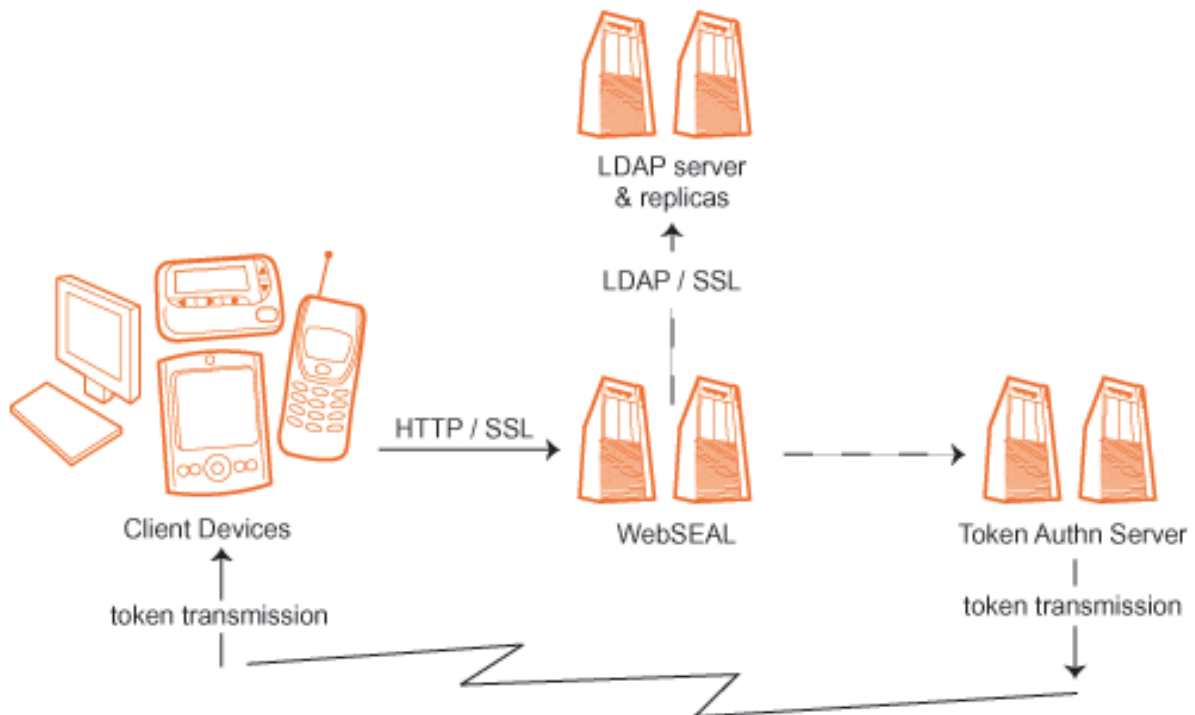
- Something that they *have*, such as a mobile phone, a token, a pager, or PDA.

Multiphased authentication architecture

Let's outline the high-level architecture we'll use for implementation of our sample two-factor authentication system. In such a system, there exist a number of key components:

- **Authentication/authorization system:** Drives the authentication process. In this tutorial, we'll use TAM WebSEAL.
- **One-time passcode generator:** Generates the one-time passcodes.
- **Transport layer:** Transports the one-time passcode to the end user.
- **Passcode receiver:** The device or application that receives the one-time passcode delivered by the transport layer.

The architecture for a TAM-centric multiphased authentication system is illustrated in the following figure, with each of the components highlighted.



In this tutorial, we present an implementation of a token cross-domain authentication service (CDAS) that handles the two-factor authentication. The sections that follow outline the mechanism for building such a system. The next section introduces the

CDAS interface used within WebSEAL to implement two-phased authentication.

Section 3. Cross-domain authentication service (CDAS)

The CDAS interface

The CDAS interface provided by TAM Web components allows for authentication and mapping of end users to third-party authentication providers. There are a number of interfaces available to the customer within this documented API. These include:

- **Token authentication:** Allows for implementation of code that will support token authentication systems.
- **Username/password authentication:** Allows for the implementation of code that will support authentication of usernames and passwords.
- **Certificate authentication:** Allows for the implementation of code that will support authentication of certificates. It can be used to map end users with certificates to users defined within the TAM LDAP repository.

CDAS functions

For each implementation of the CDAS, there are four functions that must be implemented:

1. `xauthn_initialize`: Initializes the interface and is called once.

```
xauthn_status_t
xauthn_initialize(
    int      argc,      /* in */
    const char **argv  /* in */
)
{
}
```

2. `xauthn_shutdown`: Shuts down the WebSEAL server.

```
xauthn_status_t
```

```
xauthn_shutdown(  
int      argc,      /* in */  
const char **argv  /* in */  
)  
{  
}
```

3. xauthn_authenticate: Performs authentication.

```
xauthn_status_t  
xauthn_authenticate(  
    xnvlst_t      *authnInfo,  
    xauthn_identity_t *ident  
)  
{  
}
```

4. xauthn_change_password: Changes the user password.

```
xauthn_status_t  
xauthn_change_password(  
    xnvlst_t      *authnInfo  
)  
{  
}
```

Example implementations of CDAS code are provided within the Access Manager WebADK package of TAM. That code is a good place to start to understand the implementation specifics of such a module. The following sections describe an implementation of a CDAS for solving the multiphase authentication problem.

Section 4. WebSEAL token CDAS interface

Overview

WebSEAL provides an interface for development of authentication to third-party token providers. This section outlines a process for providing multiphase authentication capabilities using the WebSEAL token interface. It uses the next-token authentication status to force a multiphase authentication process. This section does not outline the implementation of solutions in which multiphased

authentication contains real-token implementations requiring next-token. However, you could potentially write logic within this application that allows for such situations.

The WebSEAL interface implements the standard CDAS interfaces described in [The CDAS interface](#). The default out-of-the-box implementation of the token CDAS supports the SecureID token authentication mechanism, which communicates with the RSA ACE server.

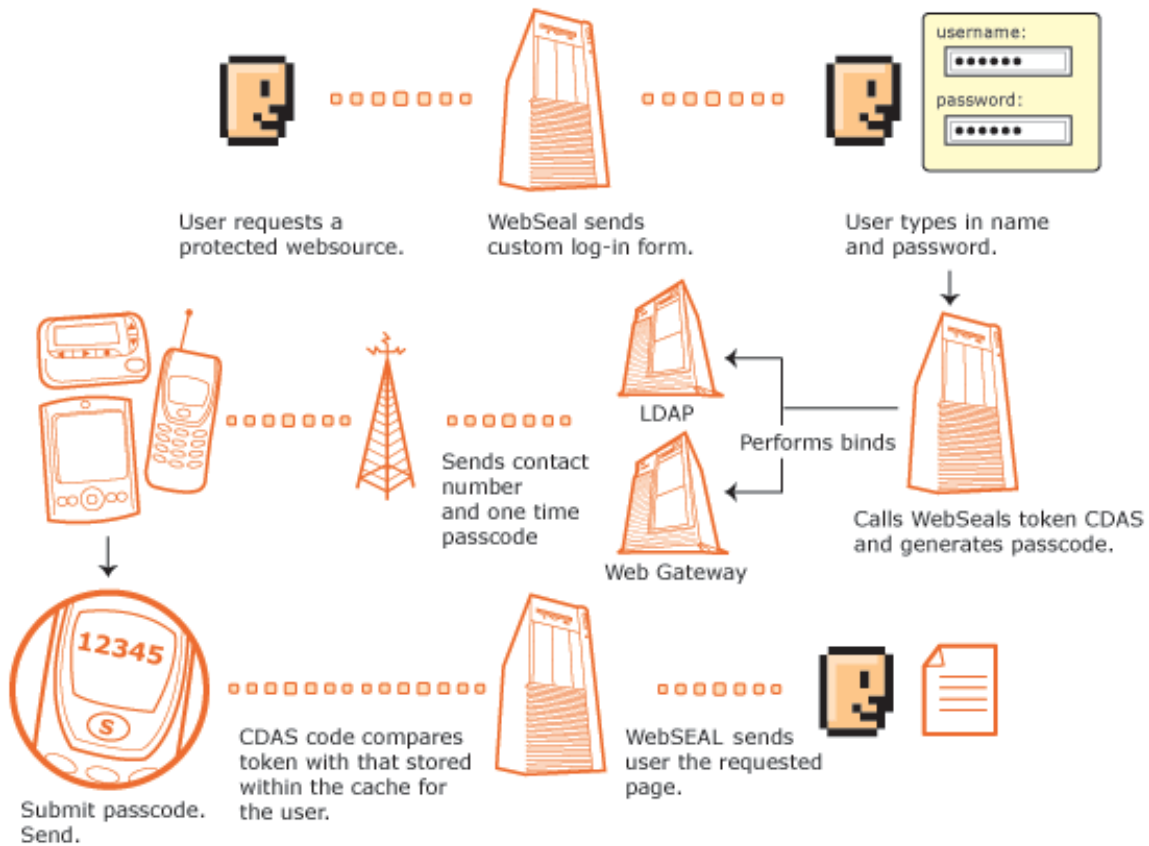
The following section outline a multiphased authentication scheme that makes use of SMS (Short Message Service) mobile phone technology. This technology provides the ability to supply a one-time token via an SMS message to a user's mobile phone.

Token CDAS process flow

Let's look at the process flow of the token CDAS solution described previously. First, our components:

1. **Authentication/authorization system:** WebSEAL is used as the driver for authentication and authorization.
2. **Passcode generator:** WebSEAL CDAS code generates a one-time passcode. This passcode could be stored within WebSEAL memory or in an LDAP directory.
3. **Transport layer:** The mobile phone network.
4. **Passcode receiver:** The mobile phone (with SMS being the mechanism for transportation).

The following figure illustrates the WebSEAL token CDAS high-level workflow.



[Click here](#) to see an animated version of the workflow.

The process flow is as follows:

1. The user requests a protected Web resource via WebSEAL (let's assume it's the file `index.html`).
2. WebSEAL determines if the Web user needs to be authenticated and sends a custom login form. This form is actually a customization of WebSEAL's `pkmslogin.html` (i.e., `PDWeb/www/lib/html/C/tokenlogin.html`). `tokenlogin.html` contains input fields for a user ID and password. The form action is directed to the standard `pkmslogin.form`.
3. The user enters the appropriate user ID and password and submits them to WebSEAL.
4. WebSEAL's token CDAS interface is called. Authentication is also performed using `xauthn_authenticate` (see [Authenticating the user](#)). On startup, the CDAS code should perform any binds that are required

within the CDAS lifetime. This includes binding to the LDAP server and the Web gateway. When the `POST` data is received from the user:

1. CDAS code authenticates the username and password with LDAP. This can be performed through a call to `azn_authenticate()` (for details, see documentation on the TAM authorization API), or can simply be a call directly to the LDAP server.
2. CDAS code retrieves the contact number of the end user authenticated from LDAP.
3. CDAS code generates a one-time passcode to be stored within the CDAS cache.
4. CDAS code sends a message to the SMS gateway for the end user to receive the one-time passcode.
5. WebSEAL returns the `nexttoken.html` page, which the end user uses to key in the passcode once it has been received through the mobile phone.
5. The SMS gateway sends the contact number and the one-time passcode to the mobile network.
6. The mobile network delivers the one-time passcode to the contact number.
7. Upon receiving the passcode via the mobile device, the user keys it in and submits the form to WebSEAL's `pkmslogin.form` again.
8. The CDAS code compares the token with the one stored within the cache for the end user. Upon success, the end user is returned the originally requested page -- `index.html`, in this case.

Section 5. Multiphased authentication CDAS code

Global variables

The following variables are used in the code examples that follow.

```
#include <winhttp.h>

#define AUTHNMECH_INFO "Token CDAS"

char **nextTokenModeList; /* This is the list of username and one-time code */
static int counter; /* length of the list */
static int oneTimePassword; /* The integer used to increment value for the
                             one-time passcode*/
```

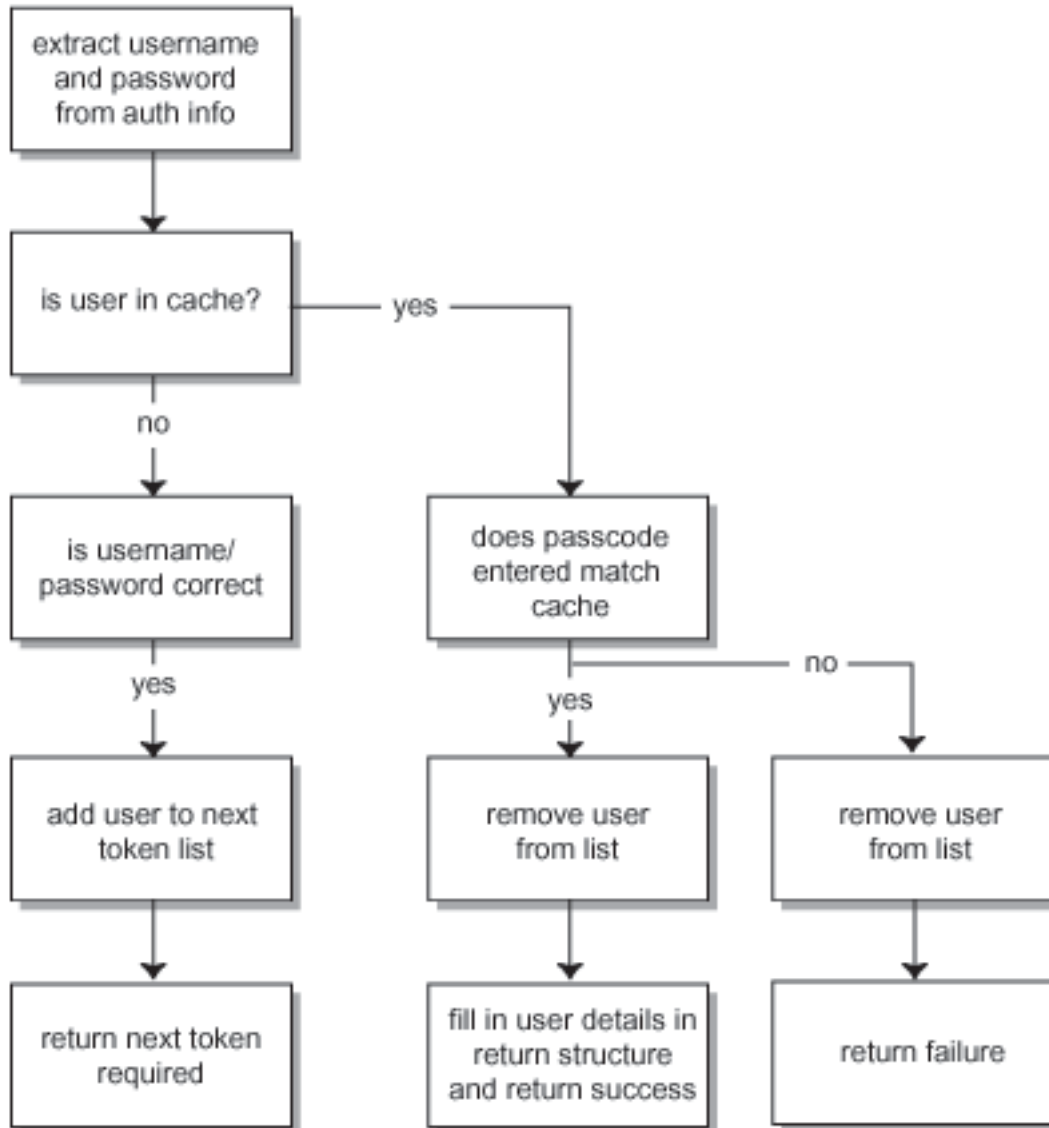
Initializing the CDAS interface

This interface, `xauthn_initialize()`, is called the first time an authentication is performed. It is called once only for the lifetime of the WebSEAL execution.

```
xauthn_status_t
xauthn_initialize(
    int      argc,      /* in */
    const char **argv  /* in */
)
{
    counter = 0;
    nextTokenModeList = malloc(sizeof(char **));
    oneTimePassword = 0;
    return XAUTHN_S_COMPLETE;
}
```

Authenticating the user

The flow of the code for `xauthn_authenticate()` is as follows:



There are a few instances in which we've simplified the sample's logic:

- The sample code does not perform the "Is username and password correct?" authentication step shown in the figure above. This step has been left as an exercise for the reader.
- The example code also does not perform the lookup to the LDAP registry for the user's mobile phone number; for simplicity's sake, the number is hardcoded into the example.
- The example code does not mutex the counter flag to ensure that multiple threads do not access the counter at the same time. You'll have to implement such functionality yourself if you want it. See the documentation at the Microsoft Developers Network for more information

(see [Resources](#)).

```

xauthn_status_t
xauthn_authenticate(
    xnvlst_t          *authnInfo,
    xauthn_identity_t *ident
)
{
    int          i;
    char         *username;
    char         *password;
    char         *error_val;
    char         **name;
    int          prin_type;
    char         prin_type_str[128];
    azn_creds_h_t creds;
    xnvlst_item_t *item;
    xauthn_status_t st;
    azn_string_t  prin_name = NULL;
    int           found;
    int           count;
    int           moveVar;
    char         *nextTokenUser;

    st = XAUTHN_S_NEXT_TOKEN;
    password = NULL;
    username = NULL;

    /* Lets traverse the authnInfo passed in to get their user information */
    if ( authnInfo != NULL ) {
        for (i=0; i<authnInfo-> length; i++) {
            item = &authnInfo->items[i];
            if ( item->name != NULL) {
                if (!(strcmp(item->name, "xauthn_username"))) {
                    username = strdup(item->value);
                } else if (!(strcmp(item->name, "xauthn_token"))) {
                    password = strdup(item->value);
                }
                printf("%s:\t%s (%d)\n",
                    item->name,
                    item->value,
                    item->vlen);
            }
        }
    }

    /*
    This code looks through the simple list and determines if the user
    is in the state of supplying their token information. Found indicates
    whether the user is in the process of supplying SMS information
    */

    found = FALSE;

    for (count=0; count<counter; count++) {
        if (!(found)) {
            /* lets just look through the list */
            if (nextTokenModeList[count] != NULL) {
                /* Does this entry contain the username ? */
                if (strstr(nextTokenModeList[count], username) != NULL) {
                    /* They are in next Token Mode */
                    free(nextTokenModeList[count]);
                    /* move the list forward */
                    for (moveVar=0; moveVar<counter; moveVar++) {

```

```

        nextTokenModeList[count+moveVar] =
            nextTokenModeList[count+moveVar+1];
    }
    nextTokenModeList[counter] = NULL;
    counter--;
    st = XAUTHN_S_COMPLETE;
    found = TRUE;
    }
    }
}

if (!(found)) {
    /*
    If they are not found add them to the nextTokenModeList
    Firstly authenticate the user...
    azn_util_password_authenticate() can be used here It is
    assumed in this code that the user is supplying a correct user
    id and password It is left as an exercise for the user to
    implement this.
    */
    oneTimePassword++;
    /*
    Generate a bogus one time passcode: Code needs to be added to
    here to generate a guaranteed unique one-time passcode
    */
    nextTokenUser = malloc((strlen(username) + 2)*(sizeof(char)));
    sprintf(nextTokenUser,
        "%s%d",
        username,
        oneTimePassword);
    nextTokenModeList[counter] = strdup(nextTokenUser);
    counter++;
    free(nextTokenUser);
}

/* Indicate LDAP identity type */
ident->prin.prin_type = XAUTHN_PRIN_TYPE_DN;
ident->prin.data.dn = (char *)strdup(username);

/* Common identity components */
ident->user_info = (char *)NULL;
ident->authnmech_info = (char *)strdup(AUTHNMECH_INFO);

/*
Now lets send the string to the SMS gateway if they are in next token
mode
*/

if (st == XAUTHN_S_NEXT_TOKEN) {
    BOOL bResults = FALSE;
    HINTERNET hSession = NULL,
    hConnect = NULL,
    hRequest = NULL;

    // Use WinHttpOpen to obtain a session handle.
    hSession = WinHttpOpen( L"A WinHTTP Example Program/1.0",
        WINHTTP_ACCESS_TYPE_DEFAULT_PROXY,
        WINHTTP_NO_PROXY_NAME,
        WINHTTP_NO_PROXY_BYPASS, 0);

    // Specify an HTTP server.
    if (hSession) {
        printf("Creating session\n");
        hConnect = WinHttpConnect( hSession, L"whatever.sms.com",
            INTERNET_DEFAULT_HTTP_PORT, 0);
    }

    // Create an HTTP Request handle.

```

```

        if (hConnect) {
            printf("Creating session\n");
            hRequest = WinHttpOpenRequest( hConnect, L"GET",
L"/uri/SmsGateway?username=tivoli&password=iloveit&message=
whatever&to=%2B6145451454&from=%2B61408432783",

                                NULL, WINHTTP_NO_REFERER,
                                WINHTTP_DEFAULT_ACCEPT_TYPES, 0);

        }

        // Send a Request.
        if (hRequest) {
            printf("Creating request\n");
            bResults = WinHttpSendRequest( hRequest,
                                WINHTTP_NO_ADDITIONAL_HEADERS, 0,
                                WINHTTP_NO_REQUEST_DATA, 0,
                                0, 0);

        }

        // Report any errors.
        if (!bResults)
            printf("Error %d has occurred.\n", GetLastError());

        // Close any open handles.
        if (hRequest) WinHttpCloseHandle(hRequest);
        if (hConnect) WinHttpCloseHandle(hConnect);
        if (hSession) WinHttpCloseHandle(hSession);
    }

    /* and return the status */
    return st;
}

```

Building the CDAS code

In this section, you'll learn how to modify the makefile to support the code in the previous section. You'll use the example CDAS `Makefile.win32` to make the necessary changes.

The following listing shows the modified makefile that comes with the example CDAS code. Replace the `xauthn.c` example code with the code in the previous section. It incorporates the use of the WinHTTP code within the module.

```

# Microsoft Developer Studio Generated NMAKE File, Based on xauthn.dsp
#
#
#
#

CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

OUTDIR=.
INTDIR=.
# Begin Custom Macros
OutDir=.
# End Custom Macros

```

```

ALL : "$(OUTDIR)\xauthn.dll"

CLEAN :
    -@erase "$(INTDIR)\vc60.idb"
    -@erase "$(INTDIR)\xauthn.obj"
    -@erase "$(OUTDIR)\xauthn.dll"
    -@erase "$(OUTDIR)\xauthn.exp"
    -@erase "$(OUTDIR)\xauthn.lib"

"$$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

CPP_PROJ=/nologo /MTd /W3 /Gm /GX /ZI /Od /I "C:\Program Files\WinHTTP 5.0 \
SDK\inc" /I "C:\Program Files\Microsoft SDK\include" \
/I "../include" /I "../..../PD/include" \
/D "WIN32" /D "NDEBUG" /D "_WINDOWS" \
/D "_MBCS" /D "_URL" /D "XAUTHN_EXPORTS" \
/Fp"$$(INTDIR)\xauthn.pch" /YX /Fo"$$(INTDIR)\\" \
/Fd"$$(INTDIR)\\" /FD /c

MTL_PROJ=/nologo /D "NDEBUG" /mktyplib203 /win32
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$$(OUTDIR)\xauthn.bsc"
BSC32_SABERS= \

LINK32=link.exe
LINK32_FLAGS="..\..\PD/lib/libatrc.lib" \
"C:\Program Files\WinHTTP 5.0 SDK\lib\winhttp5.lib" \
../lib/pdxauthn.lib ../lib/pdpthread.lib ../lib/ldap.lib \
../lib/pdauthn.lib ../lib/pdxauthnutils.lib \
"..\..\PD/lib/pdauthzn.lib" msvcr.lib ws2_32.lib \
/nologo /dll /NODEFAULTLIB:LIBCMT.lib /NODEFAULTLIB:LIBCMD.lib \
/pdb:"$(OUTDIR)\xauthn.pdb" /machine:I386 \
/out:"$(OUTDIR)\xauthn.dll" /implib:"$(OUTDIR)\xauthn.lib"

LINK32_ORBS= \
    "$(INTDIR)\xauthn.obj"

"$$(OUTDIR)\xauthn.dll" : "$(OUTDIR)" $(DE_FILE) \
    $(LINK32_ORBS)
    $(LINK32) @<<
    $(LINK32_FLAGS) $(LINK32_ORBS)
<<

SOURCE=xauthn.c

"$$(INTDIR)\xauthn.obj" : $(SOURCE) "$(INTDIR)"
    $(CPP) $(CPP_PROJ) $(SOURCE)

```

Configuring WebSEAL to support the CDAS module

To configure the newly created CDAS module:

1. Build the newly created CDAS code using the makefile in the previous section. Issue the following command within the appropriate directory:

```
NMAKE -faa Makefile.win32
```

2. Copy `xauthn.dll` to the WebSEAL `bin` directory.
3. Modify the WebSEAL configuration. This entails changes to the `webseald.conf` file. Add the following to the authentication mechanisms stanza:

```
[authentication-mechanisms]
token-cdas = xauthn.dll
```

4. Modify the authentication schemes to be used:

```
[token]
token-aut = http
```

5. Restart WebSEAL so these changes take effect.
6. Test the configuration:
 1. When testing this configuration (by connecting to WebSEAL through HTTP) make sure the correct login fields are filled in by the test user. The fields are shown in the figure below as part of the standard token login form.

Token Authentication

• Username

• Password

2. Enter a username and password for the first authentication form, followed by the number corresponding to the counter within the CDAS code when the next token form is displayed, as shown in the figure below. For the first authentication attempt, this will be the numeral 1, and is incremented (as per the code) by 1 with each new user authentication.

Access Manager for e-business Login - Next Token Required

• Next Token

Login

7. Make sure all shared libraries are placed within the system path in order for the solution to function. This includes the WinHTTP shared libraries.
-

Section 6. Summary

The solution presented here was an implementation of a multiphased authentication system based on generic hardware. This allows an organization to leverage existing hardware carried by the majority of its customers to provide multiphased authentication functionality.

WebSEAL provides a CDAS interface that can be readily used to support such an authentication scheme. The example code in this tutorial prototypes this ability. It should be noted that this implementation could also be applied to the Web plug in technology developed as part of the Access Manager Web solutions.

After taking this tutorial, you should be familiar with the implementation of such a system, and be able to implement basic CDAS code for other purposes.

Resources

Learn

- The following resources were used in the development of the examples for this tutorial:
 - [Microsoft MSDN for WinHTTP code](#)
 - Microsoft Development Environment -- see [Microsoft Developers Network](#) for details.
 - Access Manager WebSEAL C Developers Guide, which is provided as part of the Access Manager distribution.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- To develop a prototype for this tutorial, you'll need to download the following software:
 - IBM Tivoli Access Manager Base V4.1 for Windows (GA; P129): This package includes the IBM Directory server.
 - IBM Tivoli Access Manager Web Security V4.1 for Windows (E134): This package includes the WebSEAL component as well as the development API.
 - IBM Tivoli Access Manager Web Portal Manager (Graphical Administration Tool) V4.1 for Windows (GA; P133): This component is not required, but enables for administration of Access Manager from a Web browser.
- These components can be installed on a single server, or on multiple servers.
 - You'll need to apply fix pack patches to the infrastructure for example in the tutorial to function as expected. You can download these patches from [IBM Tivoli Access Manager for e-business](#). Fix packs are required for both the WebSEAL component and the Base component.
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

Discuss

- [Participate in the discussion forum for this content](#).

About the author

Christopher Hockings

Christopher Hockings is a member of the Advanced Customer Engineering team working in the Tivoli Security Business Unit (part of the IBM Software Group). He specializes in providing architecture and integration solutions for customers using the Tivoli Access Manager product suite. This includes building specialized development modules for customers based on the Access Manager product suite. Chris was a member of the DASCOT team when it was acquired by IBM. He has attained a bachelor's degree in engineering and bachelor's degree in information technology from Queensland University of Technology.