Two-Factor Authentication using Tivoli Access Manager WebSEAL

Christopher Hockings

October 06, 2005
(First published May 01, 2003)

This paper focuses on the implementation of two-factor authentication methods using the Tivoli Access Manager (TAM) WebSEAL product. It aims to provide the reader with an overview of two-factor authentication systems supported within TAM WebSEAL, and presents options for extending the capabilities to include other two-factor authentication systems. It documents the method of implementation of a WebSEAL cross-domain-authentication-service (CDAS) and presents a practical example of implementation using mobile phone Short Message Service (SMS). Within this example, two different configuration options are presented for WebSEAL to support the service, one that implements a token CDAS and another that uses the simple username/password CDAS. In each of these implementations, the advantages and disadvantages of each are presented.

Introduction

Single factor authentication such as password authentication is in many cases no longer considered as secure in the Internet world as users like to use "easy-to-remember" and hence "easy-to-guess" passwords, such as date of birth, children or family members’ name, pet's name, 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. Strong authentication is based upon the user being required to present authentication identifiers (or credentials) based on two or more factors. Traditionally, this requires the end user 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.

This paper presents two solutions for implementing two phased authentication mechanisms within WebSEAL:

1. The implementation of a token CDAS interface which uses a one-time password (OTP) generator to store authentication information pushed to the delivery device, where the delivery device in our example is a mobile phone device.
2. The implementation of a stand-alone application (servlet or CGI) that uses the OTP generator to generate a one-time password to be stored for a particular authenticated user.

High-Level Assumptions

In order to deploy one-time generated pass-code delivery via SMS, there are a number of requirements of the users of such a system:

- Guaranteed delivery time: The time for delivery of the one-time generated pass-code with SMS varies in different situations. It is assumed throughout this document that this is within 20 seconds (which is normally the case).
- A user must be within a mobile phone operator's coverage: The mobile phone that is used to receive the one-time generated pass-code must be within the mobile phone operator's coverage or results in lost of delivery.
- The user must not be a frequent international traveler: frequent travelers between different countries may suffer from lost of one-time generated pass-code delivery because the support of SMS varies from different network providers in different countries.

The remainder of this document assumes that the user satisfies the requirements above.

High-Level Overview

This section outlines the high level architecture to be used for implementation of two-factor authentication. In such a system, there exists a number of key components:

1. Authentication/Authorization system: Throughout this paper this is assumed to be TAM WebSEAL. This drives the authentication process.
2. One-Time Pass-code generator: This generates the one-time pass-codes.
3. Transport Layer: This transports the one-time pass-code to the end user.
4. Pass-code receiver: This is the device/application that receives the one-time pass-code delivered by the transport layer.

The architecture for such a system would appear as follows, with each of the components highlighted.
In the figure above, it is possible to build a two-factor authentication system using the following options:

1. WebSEAL with Token Authentication configured: This requires a token CDAS to be written that handles the two-factor authentication. Sections that follow outline the mechanism for building a token CDAS to support two-phased authentication.
2. WebSEAL and External Authentication Server: This assumes that the external authentication server offers a service that will generate one-time authentication tokens for use in a two-phased authentication system. It is assumed also that this system would authenticate this token. This relies on the implementer to build a secure one-time pass-code generation utility.

This document outlines the implementation solutions for each of these and presents example code for implementing such systems on a TAM infrastructure.

**CDAS interface**

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

1. Token Authentication: This allows for implementation of code that will support token authentication systems.
2. Username/password authentication: This allows for the implementation of code that will support authentication or username and password.
3. Certificate Authentication: This allows for the implementation of code that will support authentication of certificates. It can be used to map users with certificates to users defined within the TAM LDAP repository.

Example implementations of CDAS code are provided within the Access Manager for Web ADK package of TAM. This code is a good place to start to understand the implementation specifics.
of such a module. The following sections describe implementations of 1 and 2 as alternatives for solving the multiphase authentication solutions above.

**WebSEAL and Token CDAS**

WebSEAL provides an interface for development of authentication to third-party token providers. This section outlines a process for providing multiphase authentication capabilities using this token interface. It uses the next-token authentication status to force a multi-phase authentication process. With this in mind, this section does not outline implementation of solutions where multiphased authentication contains real-token implementations requiring next-token. However, this does not infer that smarts cannot be written within this application that allow for such situations.

The components are described as follows:

1. **Authentication/Authorization system**: WebSEAL is used as the driver for authentication and authorization.
2. **Pass-code generator**: WebSEAL CDAS code performs the generation of a one-time pass code. This pass code could be stored within WebSEAL memory or LDAP.
3. **Transport Layer**: This is the mobile phone network.
4. **Pass-code receiver**: This is the mobile phone (with SMS being the mechanism for transportation).

**Figure 2: WebSEAL and Extended LDAP high-level workflow**

The process flow is as follows:

1. User requests a protected web resource via WebSEAL (let’s assume index.html).
2. WebSEAL determines if the user needs to be authenticated and sends a custom login form. This custom login form is actually a customization from WebSEAL’s pkmslogin.html (i.e. PDWeb/www/lib/html/C/tokenlogin.html) The tokenlogin.html contains two input fields for the user ID and password. The form action is directed to the standard pkmslogin.form.
3. User inputs the user id and password and submits to WebSEAL.

4. WebSEAL’s token CDAS interface is called. 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:
   4.1. 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.
   4.2. CDAS code retrieves the contact number of the user authenticated from LDAP.
   & 4.3. CDAS code generates a one-time passcode to be stored within the CDAS cache.
   4.4. CDAS code sends message to SMS gateway for end user to receive one time passcode.
   4.5. WebSEAL returns the nexttoken.html page which the user uses to key in the passcode, upon receipt through the mobile phone.

5. SMS gateway sends the contact number & the one-time pass-code to the Mobile Network.

6. The Mobile Network delivers the one-time pass-code to the contact number.

7. User keys in the one-time pass-code (upon received from the mobile device) and sends the form to WebSEAL pkmslogin.form again.

8. CDAS code compares the token with that stored within the cache for the user. Upon success, the user is returned the original requested page, in this case, index.html.

**Benefits of this model**

- Central user repository with LDAP.
- All processing is performed within the WebSEAL memory and no callouts are made to other applications (other than potentially LDAP).
- Performance of cache solution high.

**Implementation and Architectural Considerations**

Architecturally, this solution is preferred; given there is no reliance on external applications, thereby reducing risk and error states. There are a number of considerations, however, if considering this approach:

- No write operations required to LDAP server.
- Careful implementation of multi-threaded cache code needs to be considered when avoiding the write to LDAP.
- When using with token authentication, extra considerations need to be made given this solution re-uses the next token mode within the process of token authentication.
- Cache will not be shared across WebSEAL servers within the environment. Load balancers need to ensure sticky-bit is active.
- One-time passcode is lost in cache when WebSEAL re-cycles.

In this CDAS, the flow requires that a CDAS cache keep track of the generated pass codes. The alternative is to write this data to LDAP as per the solution below. Writing the data to LDAP is certainly a simpler method of implementation, given the multi-threaded issues that are involved within such a cache implementation, however, the performance of the cache implementation would greatly improve on that of the LDAP implementation. Writing data to LDAP is a costly operation, so this needs to be considered when implementing a production ready system such as this.
Analysis
This solution requires the developer to construct suitable code for use within the WebSEAL multi-threaded environment where mutual exclusion techniques are required when implementing the CDAS cache. However, the performance of such code will benefit the implementation. It also avoids communication with outside servers for pass-code generation or storage between authentication phases. Hence it limits the number of race conditions possible when considering lags within the design.

The appendix outlines some example code, which can be used to simulate the next token mode described within this solution. Also within the appendix is outlined example WinHTTP code for communication with an SMS gateway. It is a requirement that this be installed when building the code. Also, the libraries for building the code need to be included within the test environment.

WebSEAL and External Authentication Server

This solution makes use of the standard userid/password CDAS implementation. The implementation makes use of external servers for generation and maintenance of the pass-codes. It uses a OTP generator for generation of pass-codes, LDAP for storage of a sequence number (as the OTP seed) between transactions, as well as CGI code to instantiate the OTP generation process.

The components are described as follows:

1. Authentication/Authorization system: WebSEAL is used as the driver for authentication and authorization.
2. Pass-code generator: A custom CGI application is built for the purpose of the random one-time pass-code generation.
3. Transport Layer: This is the mobile phone network.
4. Pass-code receiver: This is the mobile phone (with SMS being the mechanism for transportation).

The OTP generator is used to:

- Retrieve user's contact number and a sequence number from the LDAP server
- Generate a random one-time pass-code with the sequence number
- Return a custom second login form to user which contains the user ID, input fields for the secret PIN and the random one-time pass-code, and the form action as WebSEAL pkmslogin.form
- Send the generated random one-time pass-code & contact number to SMS gateway

The custom CDAS is used to verify the one-time pass-code that users receives via the local mobile phone network and to authenticate users against the LDAP server in the Access Manager secure domain. Users are required to authenticate themselves with a secret PIN, and a one-time generated pass-code that they received from the mobile network.
Figure 3: WebSEAL and Extended LDAP high-level workflow

1. User requests a protected web resource via WebSEAL.
2. WebSEAL determines if the user needs to be authenticated and sends a custom login form. This custom login form is actually a customization from WebSEAL's pkmslogin.html (i.e. PDWeb/www/lib/html/C/login.html). The login.html contains only one input field for the user ID. The form action is directed to a custom CGI application (the one-time pass-code generator).
3. User inputs the user ID and the form is sent to the One-Time Pass-code generator.
4. When the OTP generator starts up, it retrieves the shared secret PIN from the LDAP server over a secure tunnel with the built-in LDAP query function. This shared secret PIN is one of the components to be used to generate the one-time pass-code.
   a. Once the OTP generator is called with the user ID, it firstly retrieves the contact number and a sequence number according to the user ID from the LDAP server over a secure connection with the built-in LDAP query function.
   b. It then generates the random one-time pass-code with the sequence number, and the shared secret PIN.
   c. Upon successful one-time pass-code generation, the OTP generator sends a second login form to the user to request the user's secret PIN and the one-time pass-code. This second login form also contains the user ID and the form action is directed to WebSEAL pkmslogin.form
   d. At the same time, the OTP generator also sends the one-time pass-code and the contact number to the SMS gateway.
5. SMS gateway sends the contact number & the one-time pass-code to the Mobile Network.
6. The Mobile Network delivers the one-time pass-code to the contact number.
7. User keys in the secret PIN and the one-time pass-code (upon received from the mobile device) and sends the form to WebSEAL pkmslogin.form
8. The custom CDAS acts as the authentication engine. When WebSEAL starts up, the custom CDAS retrieves the shared secret PIN from the LDAP server over a secure channel with built-
in LDAP query function. This shared secret PIN is shared between the custom CDAS and the OTP generator for the one-time pass-code generation and verification.

a. The custom CDAS is invoked after user input the secret PIN & one-time pass-code. Firstly, the custom CDAS retrieves the sequence number according to the user ID from the LDAP server over a secure connection with the built-in LDAP query function.
b. It generates the random one-time pass-code with the sequence number, and the shared secret PIN. Then, it compares the random one-time pass-code with the one that user input.
c. If the one-time pass-code is matched, the custom CDAS then authenticate users with the LDAP server. It also updates the sequence number by 1.

Benefits of this model

- Central user repository for storage of authentication data.
- No need for complex CDAS caching code.
- Provides disjoint implementation, which can avoid race conditions and load balancer error.

Implementation and Architecture Considerations

- Requires extension of LDAP schema to include pass-code field.
- All LDAP servers (master and replicas) require read and write operation for performance, which is not supported in all TAM supported LDAP repositories.
- Use of CGI code would introduce a bottleneck in the authentication flow.
- LDAP replication delay should be considered (although this can be considered less of a problem given the SMS mobile message needs to make it to the phone).

Conclusion

When users log into a WebSEAL-protected website, the custom token CDAS will automatically deliver them a secure, one-time generated pass-code with SMS. This allows an organization to build a strong authentication security at a budget price.

Users get instant benefit if they already carry a phone, they don't need any additional device to receive the one-time generated pass-code (zero client footprint). There is no hardware cost, so the deployment costs are kept to minimal. Security has been significantly improved compared with one-factor (weak) authentication.

The solutions presented above give alternatives for implementation of a solution for multi-phased authentication. These solutions have their merits, and these implementations should be used as a reference when building such systems in a customer's environment.

Appendix A: Example source code for token CDAS solution

This section contains source code for implementation of the token authentication module, we well as source code that can be used for communicating with a Forms Based SMS gateway. The configuration of WebSEAL needs token-auth to be turned on with the library generated from the code below.

The following code snippet indicates the process by which the CDAS code should be written. It does not incorporate any thread safe caching, but merely uses an ordered list to indicate users
that have authenticated as well as their one-time pass-code. The code also uses the winHTTP API that is part of MSDN. This code is used to communicate with an SMS gateway for sending the one-time pass-code to a mobile device. This code has undergone unit testing with a real SMS gateway.

/

char **nextTokenModeList;
static int counter;
static int oneTimePassword;
/

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

xauthn_status_t
xauthn_authenticate(
    xnvlist_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;
    xnvlist_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 they are not found add them to the nextTokenModeList */
  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 let's 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=%2B61402893662&from=%2B61402855469",
            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;