**Sample Code**

This section lists a sample code that will help you get started in using the ArubaOS XML API interface. These codes have been tested in a controlled environment. We recommend that you test this code in a non-production environment before using it for actual user management tasks.

**Using XML API in C Language**

The example script is written in the *C* language. The example script (auth.c) sends an authentication request from your authentication server to the controller.

|  |  |
| --- | --- |
|   | This is an example code and is provided for illustration purposes. If you plan to use this code in your environment, ensure that the code meets your IT guidelines. Also create an error free executable to successfully execute the script. |

**Figure 1***Authentication Script Listing*

##### *auth.c listing*

##### Authentication Script Example -- Start --

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <getopt.h>

char \*command, \*ipaddr, \*macaddr;

char \*name, \*password, \*role;

char \*tout, \*secret;

char \*auth, \*key, enchashbuf[41];

unsigned char hashbuf[20];

char \*version;

char post[4096], cmdbuf[512], encbuf[1024];

#define DEBUG

#ifdef DEBUG

#define debug(x...) fprintf(stderr, x)

#else

#define debug(x...)

#endif

extern int cgi\_escape\_url(char \*t, int tl, char \*s, int sl, int b\_newline);

static void encode\_message\_digest (unsigned char \*md, int mdlen, char \*output);

static void usage (void)

{

 fprintf(stderr, "Usage: ecp [options] <switch> <command> [<secret>]\n");

 fprintf(stderr, " \n");

 fprintf(stderr, " <switch> Switch IP address.\n");

 fprintf(stderr, " <command> One of add, del, or authenticate.\n");

 fprintf(stderr, " <secret> Shared secret.\n");

 fprintf(stderr, " \n");

 fprintf(stderr, " -i ipaddr User IP address in A.B.C.D format.\n");

 fprintf(stderr, " -m macaddr User MAC address in aa:bb:cc:dd:ee:ff format.\n");

 fprintf(stderr, " -n name User name.\n");

 fprintf(stderr, " -p passwd User password.\n");

 fprintf(stderr, " -r role User role.\n");

 fprintf(stderr, " -t timeout User session timeout.\n");

 fprintf(stderr, " -v version API version number. Default is 1.0\n");

 fprintf(stderr, " -a method one of md5, sha-1 or cleartext.\n");

 exit(1);

}

main(int argc, char \*\*argv)

{

 char c, \*p;

 int fd, len, postlen;

 struct sockaddr\_in sa;

 while ((c = getopt(argc, argv, "a:i:m:n:p:r:t:v:")) != EOF) switch(c) {

 case 'i': /\* ipaddr \*/

 ipaddr = optarg;

 break;

 case 'm': /\* macaddr \*/

 macaddr = optarg;

 break;

 case 'n': /\* name \*/

 name = optarg;

 break;

 case 'p': /\* password \*/

 password = optarg;

 break;

 case 'r': /\* role \*/

 role = optarg;

 break;

 case 't': /\* session timeout \*/

 tout = optarg;

 break;

 case 'v': /\* version \*/

 version = optarg;

 break;

 case 'a': /\* authentication \*/

 auth = optarg;

 if (!strcasecmp(auth, "sha-1") &&

 !strcasecmp(auth, "md5"))

 usage();

 break;

 default:

 usage();

 break;

 }

 argc -= (optind - 1);

 argv += (optind - 1);

 if ((argc < 3)) {

 usage();

 }

 if (version == NULL)

 version = "1.0";

 debug("server=%s, command=%s, version=%s, secret=%s\n",

 argv[1], argv[2], version, argv[3]?argv[3]:"<>");

 if (argv[3]) secret = argv[3];

 p = cmdbuf;

 sprintf(p, "xml=<aruba command=‘%s’>", argv[2]);

 p += strlen(p);

 if (ipaddr) {

 sprintf(p, "<ipaddr>%s</ipaddr>", ipaddr);

 p += strlen(p);

 }

 if (macaddr) {

 sprintf(p, "<macaddr>%s</macaddr>", macaddr);

 p += strlen(p);

 }

 if (name) {

 sprintf(p, "<name>%s</name>", name);

 p += strlen(p);

 }

 if (password) {

 sprintf(p, "<password>%s</password>", password);

 p += strlen(p);

 }

 if (role) {

 sprintf(p, "<role>%s</role>", role);

 p += strlen(p);

 }

 if (tout) {

 sprintf(p, "<session timeout>%s</session timeout>", tout);

 p += strlen(p);

 }

 if (secret) {

 if (auth == NULL) {

 key = secret;

 auth = "cleartext";

#ifndef OPENSSL\_NO\_SHA1

 } else if (!strcasecmp(auth, "sha-1")) {

 key = enchashbuf;

 SHA1(secret, strlen(secret), hashbuf);

 encode\_message\_digest(hashbuf, 20, enchashbuf);

#endif

 } else if (!strcasecmp(auth, "md5")) {

 key = enchashbuf;

 md5\_calc(hashbuf, secret, strlen(secret));

 encode\_message\_digest(hashbuf, 16, enchashbuf);

 }

 debug("Message authentication is %s (%s)\n", auth, key);

 sprintf(p, "<authentication>%s</authentication><key>%s</key>",

 auth, key);

 p += strlen(p);

 }

 debug("\n");

 sprintf(p, "<version>%s</version>", version);

 sprintf(p, "</authresponse>");

 cgi\_escape\_url(encbuf, sizeof(encbuf), cmdbuf, strlen(cmdbuf), 0);

 postlen = sprintf(post,

 "POST /auth/command.xml HTTP/1.0\r\n"

 "User-Agent: ecp\r\n"

 "Host: %s\r\n"

 "Pragma: no-cache\r\n"

 "Content-Length: %d\r\n"

 /\* "Content-Type: application/x-www-form-urlencoded\r\n" \*/

 "Content-Type: application/xml\r\n"

 "\r\n"

 "%s",

 argv[1], strlen(encbuf), encbuf);

 inet\_aton(argv[1], &sa.sin\_addr);

 sa.sin\_family = AF\_INET;

 sa.sin\_port = htons(80);

 fd = socket(AF\_INET, SOCK\_STREAM, 0);

 if (fd < 0) {

 perror("socket");

 exit(1);

 }

 if (connect(fd, (struct sockaddr \*) &sa, sizeof(sa)) < 0) {

 perror("connect");

 exit(1);

 }

 if (write(fd, post, postlen) != postlen) {

 perror("write");

 exit(1);

 }

 while ((len = read(fd, post, sizeof(post))) > 0)

 write(1, post, len);

 close(fd);

 exit(0);

}

static void encode\_message\_digest (unsigned char \*md, int mdlen, char \*output)

{

 int i;

 for (i=0; i<mdlen; i++) {

 sprintf(output, "%02x", md[i]);

 output += 2;

 }

}

}

##### Authentication Script Example -- END --

**Understanding Request and Response**

The controller processes the authentication task and sends a response to the authentication server in the XML format to the authentication server. The XML response contains the status of the request and a code in case of an error. The example script is listed in [Figure 1](http://www.arubanetworks.com/techdocs/ArubaOS_63_Web_Help/Content/ArubaFrameStyles/XML_API/Sample_Code.htm#xml_api_1290141154_1018640).

Request format: <script\_name> [options] <controller-ip> <command> <secret\_key>

**Understanding XML API Request Parameters**

The [Table 1](http://www.arubanetworks.com/techdocs/ArubaOS_63_Web_Help/Content/ArubaFrameStyles/XML_API/Sample_Code.htm#xml_api_1290141154_1018861)list all parameter that you can use in a request.

| **Table 1:***XML API Request Parameters and Descriptions* |
| --- |
| **Parameter** | **Description** |
| script\_name  | The name of the script executable. |
| Options |

|  |  |  |
| --- | --- | --- |
|  |  | -i <ip\_addr>—Specify the client’s IP address. |

|  |  |  |
| --- | --- | --- |
|  |  | -m <mac\_addr>—Specify the client’s MAC address. |

|  |  |  |
| --- | --- | --- |
|  |  | -n <name>—Specify the client’s user name. |

|  |  |  |
| --- | --- | --- |
|  |  | -p <passwd>—Specify the client password. |

|  |  |  |
| --- | --- | --- |
|  |  | -r role—Specify the current user role of the client. |

|  |  |  |
| --- | --- | --- |
|  |  | -t timeout—User session timeout. |

|  |  |  |
| --- | --- | --- |
|  |  | -v version—API version number. Default is 1.0 |

|  |  |  |
| --- | --- | --- |
|  |  | -a method—Specify the encryption method to send the secret key. You can specify MD5 or SHA-1 or cleartext as the encryption method. By default, cleartext method is used to send the key. |

|  |  |  |
| --- | --- | --- |
|  |  | -s sessid—Active session Id |

 |
| controller-ip  | The IP address of the controller that will receive the authentication requests. |
| *command* | The authentication command sent to the controller. You can send one of the following commands per request:

|  |  |  |
| --- | --- | --- |
|  |  | add: Adds the client to your network. |

|  |  |  |
| --- | --- | --- |
|  |  | delete: Deletes the client from your network |

|  |  |  |
| --- | --- | --- |
|  |  | query: Fetches information about the client |

|  |  |  |
| --- | --- | --- |
|  |  | blacklist: Blacklists or block the client from connecting to your network |

|  |  |  |
| --- | --- | --- |
|  |  | authenticate: Authenticates the client and assigns the default authenticated role. |

 |
| secret\_key | The password used to validate the authentication request from your authentication server. See [Configuring the XML API Server](http://www.arubanetworks.com/techdocs/ArubaOS_63_Web_Help/Content/ArubaFrameStyles/XML_API/Using_the_XML_API_Server.htm#xml_api_1290141154_1018019) for more information. |

**Understanding XMl API Response**

The response message from the controller is sent in an XML format. The default format of the response is:

[Message header]

Displays the request parameters and other standard header details.

..

...

..

<response>

 <status>Status Message</status>

 <code>Code in case of an error</code>

</response>

**Adding a Client**

This command will add a client on your network.

**Figure 2***Adding a client—request and response*

john@linux:/home/john/tools/xml-api# ./auth -i 10.10.10.249 -m 00:19:d2:01:0b:aa -r logon 10.11.12.13 add $abcd$1234$

The commands sends the following information in the authentication request to the controller:

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client IP address: **10.10.10.249** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client MAC address: **00:19:d2:01:0b:aa** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication server IP address: **10.11.12.13** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication command: **add** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key to validate authentication request: **$abcd$1234$** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Verification key is sent in cleartext format |

**Response from the controller**

server=10.11.12.13, command=add, version=1.0, secret=$abcd$1234$ sessid=

Message authentication is cleartext ($abcd$1234$)

HTTP/1.1 200 OK

Date: Tue, 03 Aug 2010 23:32:16 GMT

Server:

Connection: close

Content-Type: text/xml

<authresponse>

 <status>Ok</status>

 <code>0</code>

</authresponse>

**View the updated details of the client on the controller**



**Deleting a Client**

This command will delete a client from your network.Deleting a client—request and response

john@linux:/home/john/tools/xml-api# ./auth -i 10.10.10.248 10.11.12.13 delete $abcd$1234$

This command sends the following information in the request to the controller:

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client IP address: **10.10.10.248** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication server IP address: **10.11.12.13** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication command: **delete** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key to validate authentication request: **$abcd$1234$** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key is sent in cleartext format |

**Response from the controller**

server=10.11.12.13, command=delete, version=1.0, secret=$abcd$1234$ sessid=

Message authentication is cleartext ($abcd$1234$)

HTTP/1.1 200 OK

Date: Tue, 03 Aug 2010 23:30:32 GMT

Server:

Content-Length: 56

Connection: close

Content-Type: text/xml

<authresponse>

 <status>Ok</status>

 <code>0</code>

</authresponse>

**Authenticating a Client**

This command will authenticate and change the role of a client. To illustrate the authentication command request process this section displays status of the client before and after the authentication command request.

**Status of the client before authentication**

The following show user command shows the role of the client is logon before the authentication request is processed by the controller.



The following command shows the captive portal status of the logon role of the client.

(host) (config-role) #show rights logon | include "Captive Portal profile"

 Captive Portal profile = default

**Sending the authentication command**

Use the authenticate keyword in the script to send the authentication command request.

**Figure 3***Authenticating the client—request and response*

john@linux:/home/john/tools/xml-api# ./auth -i 10.10.10.248 -n john -p password 10.11.23.24 authenticate $abcd$1234$

This commands sends the following information in the request to the controller:

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client IP address: **10.10.10.248** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client username: **john** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client password: **password** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication server IP address: **10.11.12.13** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication command: **authenticate** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key to validate authentication request: **$abcd$1234$** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key is sent in cleartext format |

**Response from the controller**

server=10.11.12.13, command=authenticate, version=1.0, secret=$abcd$1234$ sessid=

Message authentication is cleartext ($abcd$1234$)

HTTP/1.1 200 OK

Date: Tue, 03 Aug 2010 23:23:42 GMT

Server:

Connection: close

Content-Type: text/xml

<authresponse>

 <status>Ok</status>

 <code>0</code>

</authresponse>

**Status of the client after authentication**

The following show user command shows the role of the client is change to guest after the authentication request is processed by the controller.



**Querying for Client Details**

This command will fetch a all details about a client connected in your network. Querying Client Information—request and response

john@linux:/home/john/tools/xml-api# ./auth -i 10.10.10.249 10.11.12.13 query $abcd$1234$

This commands sends the following information in the request to the controller:

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client IP address: **10.10.10.249** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client username: **john** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client password: **password** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication server IP address: **10.11.12.13** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication command: **query** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key to validate authentication request: **$abcd$1234$** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key is sent in cleartext format |

**Response from the controller**

server=10.11.12.13, command=query, version=1.0, secret=$abcd$1234$ sessid=

Message authentication is cleartext ($abcd$1234$)

HTTP/1.1 200 OK

Date: Tue, 03 Aug 2010 23:34:30 GMT

Server:

Connection: close

Content-Type: text/xml

<authresponse>

 <status>Ok</status>

 <code>0</code>

 <macaddr>00:19:d2:01:0b:aa</macaddr>

 <name>john</name>

 <role>logon</role>

 <type>Wireless</type>

 <vlan>1</vlan>

 <location>N/A</location>

 <age>00:00:02</age>

 <auth\_status>Unauthenticated</auth\_status>

 <essid></essid>

 <bssid>00:00:00:00:00:00</bssid>

 <phy\_type>b</phy\_type>

 <mobility\_state>Wireless</mobility\_state>

 <in\_packets>0</in\_packets>

 <in\_octets>0</in\_octets>

 <out\_packets>0</out\_packets>

 <out\_octets>0</out\_octets>

</authresponse>

The output of the show user command displays the client information.



**Blacklisting a Client**

This command will blacklist a client and restrict it from connecting to your network. The show user-table lists the client connected on your network before processing the request to blacklist the client.



**Figure 4***Blacklisting a Client—request and response*

john@linux:/home/john/tools/xml-api# ./auth -i 10.10.10.248 10.11.12.13 blacklist $abcd$1234$

This commands sends the following information in the request to the controller:

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Client IP address: **10.10.10.248** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication server IP address: **10.11.12.13** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Authentication command: **blacklist** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key to validate authentication request: **$abcd$1234$** |

|  |  |  |
| --- | --- | --- |
|  | ⚫ | Key is sent in cleartext format |

**Response from the controller**

server=10.11.12.13, command=blacklist, version=1.0, secret=$abcd$1234$ sessid=

Message authentication is cleartext ($abcd$1234$)

HTTP/1.1 200 OK

Date: Tue, 03 Aug 2010 23:29:11 GMT

Server:

Content-Length: 56

Connection: close

Content-Type: text/xml

<authresponse>

 <status>Ok</status>

 <code>0</code>

</authresponse>

The show user-table command does not list the blacklisted client. You can use the show ap blacklist-clients command on your controller to view the list of blacklisted clients

