Chapter 35

External User Management using XML-API

This chapter introduces the ArubaOS XML API interface and briefly discusses how you can use the simple API calls to perform external user management tasks. A sample code listing at the end of the chapter to help you get started with using the XML API.

Topics in this chapter:

Overview

ArubaOS allows you to set up customized external captive portal user management using its native XML API interface. The XML API interface allows you to create and execute user management operations seamlessly on behalf of the clients or users. You can use the XML API interface to add, delete, authenticate, or query a user or a client.

How the ArubaOS XML API Works

The typical interaction between your external server and the controller happens over HTTPS post commands. A typical communication process using the XML API interface happens as follows:

1. An API command is issued from your server in XML format to the controller. The XML message or request can be composed using a language of your choice using the format described in the "XML Request" on page 718. Sample code in *C* gives a simple example. See the "Sample Code" on page 722.
2. The controller processes the XML request and sends the response to the authentication server in the XML format. The XML request is sent using HTTPS post. The common format of the HTTPS post is `https://<controller-ip>/auth/command.xml`. See "XML Request" on page 718 for more information.
3. You can use the response and take appropriate action that suit your requirements. The response from the controller is returned using predefined formats. See the "XML Response" on page 719 for more information.

Using the XML API Server

To use the XML API:

1. Configure an external XML API server
2. Associate the XML API server to an appropriate AAA profile
3. Create an XML request with the appropriate API call
4. Process XML response appropriately

NOTE

The default logon role of a client or user must have captive-portal enabled.

Configuring the XML API Server

Configure an external XML API server in your AAA infrastructure. In this example, **`10.11.12.13`** is your server. The XML API interface on the controller will receive requests from this server.

- Define the XML API server

```
(host) (config) #aaa xml-api server 10.11.12.13
```

- Specify the key used to verify requests from the your server.

```
(host) (XML API Server "10.11.12.13") #key $abcd$1234$
```

- Verify the XML API server configuration

```
(host) (config) #show aaa xml-api server
XML API Server List
-------------------
Name        References  Profile Status
----        ----------  --------------
10.11.12.13 1                          <====== Reference Count is incremented for each
usage.

Total:1
```

Associate the XML API Server to AAA profile

After you define the XML API server profile associate it to the appropriate AAA profile. If the XML API server is not correctly configured in the appropriate profile, the controller will respond with the `client not authorized` error message.

You can add XML API server references can be added to the following AAA profile depending on you requirement:

- For wireless users—Associate the XML API server to the AAA profile of the virtual AP profile.

```
(host) (config) #aaa profile wirelessusers
(host) (AAA Profile "wirelessusers") #xml-api-server 10.11.12.13
(host) (XML API Server "10.11.12.13") #key aruba123
(host) (config) #show aaa profile wirelessusers

AAA Profile "wirelessusers"
---------------------------
Parameter                           Value
---------                           -----
Initial role                        logon
MAC Authentication Profile          N/A
MAC Authentication Default Role     guest
MAC Authentication Server Group     default
802.1X Authentication Profile       N/A
802.1X Authentication Default Role  guest
802.1X Authentication Server Group  N/A
RADIUS Accounting Server Group      N/A
XML API server                      10.11.12.13
RFC 3576 server                     N/A
User derivation rules               N/A
Wired to Wireless Roaming           Enabled
SIP authentication role             N/A
```

```
(host) (config) #wlan virtual-ap wireless-vap
(host) (Virtual AP profile "wireless-vap") #aaa-profile wirelessusers
(host) (config) #show wlan virtual-ap wireless-vap

Virtual AP profile "wireless-vap"
---------------------------------
Parameter                                        Value
---------                                        -----
Virtual AP enable                                Enabled
Allowed band                                     all
AAA Profile                                      wirelessusers
802.11K Profile                                  default
SSID Profile                                     default
VLAN                                             N/A
Forward mode                                     tunnel
Deny time range                                  N/A
Mobile IP                                        Enabled
HA Discovery on-association                      Disabled
DoS Prevention                                   Disabled
Station Blacklisting                             Enabled
Blacklist Time                                   3600 sec
Dynamic Multicast Optimization (DMO)             Disabled
Dynamic Multicast Optimization (DMO) Threshold   6
Authentication Failure Blacklist Time            3600 sec
Multi Association                                Disabled
Strict Compliance                                Disabled
VLAN Mobility                                    Disabled
Remote-AP Operation                              standard
Drop Broadcast and Multicast                     Disabled
Convert Broadcast ARP requests to unicast        Disabled
Band Steering                                    Disabled
WMM Traffic Management Profile                   N/A
```

- For wired users—Asiate the XML API server to the AAA profile of the appropriate wired profile.

```
(host) (config) #aaa profile wiredusers
(host) (AAA Profile "wiredusers") #xml-api-server 10.11.12.13
(host) (AAA Profile "wiredusers") #!
(host) (config) #aaa authentication wired
(host) (Wired Authentication Profile) #profile wiredusers
(host) (Wired Authentication Profile) #show aaa authentication wired

Wired Authentication Profile
----------------------------
Parameter    Value
---------    -----
AAA Profile  wiredusers
```

- Unknown wired users—Associate the XML API server to the `default-xml-api` AAA profile.

NOTE

The `default-xml-api` AAA profile is used only to add or authenticate new users.

The following example illustrates using the `default-xml-api` AAA profile.

```
(host) (config) #aaa profile default-xml-api
(host) (AAA Profile "default-xml-api") #xml-api-server 10.11.12.13
```

```
(host) (config) #show aaa profile default-xml-api
AAA Profile "default-xml-api" (Predefined (changed))
----------------------------------------------------
Parameter                           Value
---------                           -----
Initial role                        logon
MAC Authentication Profile          N/A
MAC Authentication Default Role     guest
MAC Authentication Server Group     default
802.1X Authentication Profile       N/A
802.1X Authentication Default Role  guest
802.1X Authentication Server Group  N/A
RADIUS Accounting Server Group      N/A
XML API server                      10.11.12.13
RFC 3576 server                     N/A
User derivation rules               N/A
Wired to Wireless Roaming           Enabled
SIP authentication role             N/A
```

Your controller is now ready to receive API calls from your XML API server.

Creating an XML API Request

You can now create an XML request with an appropriate authentication command and send it to the controller via HTTPS post. The format of the URL to send the XML request is:

```
https://<controller-ip/auth/command.xml
```

In which,

- `controller-ip` is the IP address of the controller that will receive the authentication request
- `command.xml` is the XML request that contains the details of authentication.

The format of the XML API request is:

```
xml=<aruba command="<authentication_command>">
  <options>Value</options>
  ...
  <options>Value</options>
</aruba>
```

You can specify any of the following commands in the XML request:

Table 150 *XML API Authentication Command*

Authentication Command	Description
`user_add`	This command adds the user to the controllers user table.
`user_delete`	This command deletes the user from the controller
`user_authenticate`	This command will authentication the user based on the authentication rules defined in the controllers configuration.
`user_blacklist`	This command will block a user from connection to your network.
`user_query`	This command will display the current status of the user connected to your network.

The authentication command requires certain mandatory options to successfully execute the authentication tasks. The list of all available options are:

Table 151 *Authentication command options*

Options	Description	Range / Defaults
`ipaddr`	IP address of the user in A.B.C.D format.	—
`macaddr`	MAC address of the user aa:bb:cc:dd:ee:ff format.	Enter MAC address with colon.
`user`	Name of the user.	64 character string
`role`	Role name assigned after authenticating.	64 character string
`password`	The password of the user used for authentication.	—
`session_timeout`	Session time-out in minutes. User will be disconnected after this time.	—
`authentication`	Authentication method used to authenticate the message and the sender. You can use any of MD5, SHA-1 or clear text methods of authentication. This option is ignored if shared secret is not configured. It is, however, mandatory if it is configured.	—
`key`	This is the encoded SHA1/MD5 hash of shared secret or plaintext shared secret. This option is ignored if shared secret is not configured on the switch. The actual MD5/SHA-1 hash is 16/20 bytes and consists of binary data. It must be encoded as an ASCII based HEX string before sending. It must be present when the controller is configured with an xml-api key for the server. Encoded hash length is 32/40 bytes for MD5/SHA-1.	
`version`	The version of the XML API interface available in the controller. This field is mandatory is all requests.	Current version 1.0

Monitoring External Captive Portal Usage Statistics

To check the external captive portal authentication statistics use the `show aaa xml-api statistics` command. This command displays the number of times an authentication command was executed per client. The command also displays the number of times an authentication event occurred and the number of new authentication events that occurred since the last status check.

```
(host) # show aaa xml-api statistics
ECP Statistics
--------------
Statistics                             10.10.10.249
----------                             ----------
user_authenticate                      1 (0)
user_add                               1 (0)
user_delete                            1 (0)
user_blacklist                         2 (0)
unknown user                           2 (0)
unknown role                           0 (0)
unknown external agent                 0 (0)
authentication failed                  0 (0)
invalid command                        0 (0)
invalid message authentication method  0 (0)
invalid message digest                 0 (0)
```

```
Packets received from unknown clients : 0 (0)
Packets received with unknown request : 0 (0)
Requests Received/Success/Failed      : 5/3/2 (0/0/0)
```

XML Request

You can create XML requests to add, delete, authenticate, blacklist, or query a user. This section provides XML request formats that you can use for each authentication tasks:

Adding a User

This XML requests uses the `user_add` command to create a new user entry in the controllers user table. If the user entry is already present in the user table, the command will modify the entry with the values defined in the XML request.

```
xml=<aruba command="user_add">
  <ipaddr>IP-address_of_the_user</ipaddr>
  <macaddr>MAC-address_of_the_user</macaddr>
  <name>User_Name</name>
  <role>Role_Name<role>
  <session_timeout>Session_timeout</session_timeout>
  <key>Shared_Key</key>
  <authentication>MD5|SHA-1|cleartext</authentication>        #select any one
  <version>1.0</version>
</aruba>
```

The following options are mandatory when you execute the `user_add` command:

- IP Address
- Version

Deleting a User

This XML requests uses the `user_delete` command to delete an existing user from the controllers user table. If the user entry contains multiple attributes these must be specified in the XML request

```
xml=<aruba command="user_delete">
  <ipaddr>IP-address_of_the_user</ipaddr>
  <macaddr>MAC-address_of_the_user</macaddr>
  <name>User_Name</name>
  <key>Shared_Key</key>
  <authentication>MD5|SHA-1|cleartext</authentication>        #select any one
  <version>1.0</version>
</aruba>
```

The following options are mandatory when you execute the `user_add` command:

- IP Address
- Version

Authenticating a User

This XML requests uses the `user_authenticate` command to authenticate and derive a new for the user.

```
xml=<aruba command="user_authenticate">
  <ipaddr>IP-address_of_the_user</ipaddr>
  <macaddr>MAC-address_of_the_user</macaddr>
  <name>User_Name</name>
  <password>Password_for_the_user</password>
  <key>Shared_Key</key>
  <authentication>MD5|SHA-1|cleartext</authentication>        #select any one
  <version>1.0</version>
</aruba>
```

The following options are mandatory when you execute the `user_authenticate` command:

- IP Address
- Version
- Name
- Password

Blacklisting a User

This XML requests uses the user_blacklist command to blacklist a user from connecting to your network.

```
xml=<aruba command="user_blacklist">
  <ipaddr>IP-address_of_the_user</ipaddr>
  <macaddr>MAC-address_of_the_user</macaddr>
  <name>User_Name</name>
  <key>Shared_Key</key>
  <authentication>MD5|SHA-1|cleartext</authentication>        #select any one
  <version>1.0</version>
</aruba>
```

The following options are mandatory when you execute the `user_blacklist` command:

- IP Address
- Version

Querying a User Status

This XML requests uses the `user_query` command to get the status and details of a user connected to your network.

```
xml=<aruba command="user_query">
  <ipaddr>IP-address_of_the_user</ipaddr>
  <macaddr>MAC-address_of_the_user</macaddr>
  <name>User_Name</name>
  <key>Shared_Key</key>
  <authentication>MD5|SHA-1|cleartext</authentication>        #select any one
  <version>1.0</version>
</aruba>
```

The following options are mandatory when you execute the `user_blacklist` command:

- IP Address
- Version

XML Response

For every successful XML request the controller will return the processed information as an XML response. There are two types of responses: Default response and Query response.

Default Response Format

The format of a default XML response from the controller is:

```
<aruba>
  <result>Error | Ok</result>
  <code>response_code</code>
  <reason>response_message</reason>
</aruba>
```

In which,

- Result specifies if the XML result was successful or failure. If the request was successful, the result tag will contain the **`Ok`** string. If the request was a failure, the result tag will contain the **`Error`** string.
- Code is an integer number that represents the error in the request. This tag is populated only if there is an error in the request.
- Reason is message that contain descriptive information about error.

Response Codes

The following response codes are returned if the XML request return an the Error string.

Table 152 *XML Response Codes*

Code	Reason message	Description
1	`unknown user` The user specified in the XML request does not exist or is incorrect.	Returned by the user_authenticate, user_delete, user_blacklist, and user_query commands.
2	`unknown role` The specified role in the XML request does not exist in the controller.	Returned by the user_add command.
3	`unknown external agent`	Returned by all commands.
4	`authentication failed` The username and the key does not match.	Returned by commands that contain the shared_key in XML request.
5	`invalid command` The XML request contains a command not supported by ArubaOS XML API interface.	—
6	`invalid message authentication method` The authentication method specified in the XML request is not supported by the ArubaOS XML API interface.	Returned by commands that contain the authentication method in the XML request.
7	`invalid message digest`	Returned by commands that contain the shared_key in the XML request.
8	`missing message authentication` The authentication method is not specified in the XML request.	Returned by all commands that require the authentication method in the XML request.
9	`missing or invalid version number` The XML request does not contain the version number or the version number is incorrect.	Returned by all commands.
10	`internal error`	

Table 152 *XML Response Codes*

Code	Reason message	Description
11	`client not authorized` The shared key in the XML request does not match or the XML API server is not defined in the appropriate AAA profile.	Returned by all commands that require shared key to be specified in the XML request.
12	`Cant use VLAN IP`	—
13	`Invalid IP` The XML request contains invalid IP address of the user or client.	Returned by all commands that required IP address to be specified in the XML request.
14	`Cant use Switch IP` The XML request contains the controllers IP address instead of the client IP address.	Returned by all commands that required IP address to be specified in the XML request.
15	`missing MAC address` The XML request does not contain the MAC address of the user or client.	Returned by all commands that required MAC address to be specified in the XML request.

Query Command Response Format

The response of the XML request with the user_query command contains detailed information about the status of the user or client. The format of the response of a query command is:

```
<aruba>
  <result>Result</result>
  <code>Code</code>
  <reason>Reason</reason>
  <role>Role</role>
  <type>Type</type>
  <auth_status>Auth_status</auth_status>
  <auth_server>Auth_server</auth_server>
  <auth_method>Auth_method</auth_method>
  <location>Location</location>
  <age>Age</age>
  <essid>Essid</essid>
  <bssid>Bssid</bssid>
  <phy_type>Phy_type</phytype>
  <vlan>Vlan</vlan>
</aruba>
```

In which, the result, code and reason values are similar to the default response. The following responses, however, are returned only in the result code returns the **OK** string.

Table 153 *Query Response Code*

Response Code	Description
Role	Displays the current role of the authenticated user
Type	Displays is the user or client is **wired** or **wireless**.
Auth_status	Displays the authentication status of the user or client. Available values are: **authenticated** or **unauthenticated**.

Table 153 *Query Response Code*

Response Code	Description
Auth_server	Displays the name of the authentication server used for authenticating the user. This information is available only if the user is authenticated by the controller.
Auth_method	Displays the authentication mechanism used to authenticate the user. This information is available only if the user is authenticated by the controller.
Location	Displays the current location of the user / clients. For wireless clients, the location is displayed in the B.F.L format. For wired clients, the location is displayed in the slot/port format.
Age	Displays the age of user in the controller. The age is displayed in DD:HH:MM format (Day:Hours:Minutes).
ESSID	Displays the ESSID to which the user is associated.
BSSID	Displays the BSSID of the AP to which the user is associated.
Phy Type	Displays the physical connection type. One of a, b, or g.
Vlan	Displays the VLAN ID of the user.

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.

NOTE

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 186 *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 --
```

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 186 on page 722.

Request format: `<script_name> [options] <controller-ip> <command> <secret_key>`

XML API Request Parameters

The Table 154 on page 725list all parameter that you can use in a request.

Table 154 *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" on page 714 for more information.

XMI 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 187 *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

```
 (host) #show user-table

 Users
 -----
     IP              MAC              Name      Role       Age(d:h:m)  Auth  ......
 ----------    ------------        ------     ----       ----------  ----  ... [truncated]
 10.10.10.249  00:19:d2:01:0b:aa              logon      00:00:00          ......

 User Entries: 1/1
```

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 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: **`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.

```
(host) #show user

Users
-----
    IP              MAC              Name      Role      Age(d:h:m)  Auth  .....
----------    ------------        ------     ----      ----------  ----  ... [truncated]
10.10.10.248  00:19:d2:01:0b:84              logon     00:00:00          .....

User Entries: 1/1
```

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 188 *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.

```
(host) (config) #show user
Users
-----
    IP              MAC              Name    Role       Age(d:h:m)  Auth   .....
----------    ------------       ------  ----       ----------  ----   ... [truncated] .
10.10.10.248  00:19:d2:01:0b:84  John    guest      00:00:04    Web    .....

User Entries: 1/1
```

Querying Client Information

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.

```
 Users
 -----
     IP              MAC              Name     Role       Age(d:h:m)  Auth  .....
 ----------    ------------        ------   ----       ----------  ----  ... [truncated]
 10.10.10.249  00:19:d2:01:0b:aa  John     logon      00:00:01          .....

 User Entries: 1/1
```

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.

```
 Users
 -----
     IP              MAC              Name      Role       Age(d:h:m)  .....
 ----------    ------------        ------    ----       ----------  ... [truncated] ...
 10.10.10.248  00:19:d2:01:0b:84  John      guest      00:00:00    .....

 User Entries: 1/1
```

Figure 189 *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

```
 (host) (config) #show ap blacklist-clients

Blacklisted Clients
-------------------
STA                reason        block-time(sec)  remaining time(sec)
---                ------        ---------------  -------------------
00:19:d2:01:0b:84  user-defined  5                3595
```