
Barracuda Email Security Gateway API Guide

<https://campus.barracuda.com/doc/76284988/>

How the Barracuda Networks API Works

Barracuda Networks provides APIs for remote administration and configuration of the Barracuda Email Security Gateway version 7.x and above. Two sets of APIs are presented in this guide. Most of the examples shown use Perl script. Note that all API calls must use HTTP/1.1 or later.

- The **General APIs** section covers "generic" APIs that may be used with all Barracuda Networks appliances that support an API. For more information, see [General APIs](#).
- The **APIs for the Barracuda Email Security Gateway** section covers APIs that are specific only to the Barracuda Email Security Gateway. For more information, see [APIs for the Barracuda Email Security Gateway](#).

The framework of the API provides for the programmer to get or set variables inside an XML-RPC request that correspond to field values in the configuration database in the Barracuda Email Security Gateway. Some languages such as Perl, for example, provide wrappers for XML-RPC requests, providing an interface to form the request.

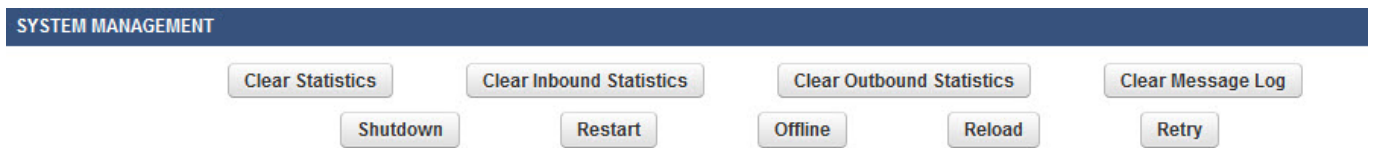
What Can Be Configured With the APIs

The APIs work through manipulation of variables inside of the system configuration database, and anything that can be declared in that database can be set or checked with the APIs. This includes most things that you can set by clicking the **Save** button in the Barracuda Email Security Gateway web interface. For example, from the **BASIC > Spam Checking** page, you can set global **Spam Scoring Limits** for the actions Block, Tag, or Quarantine, and then click **Save**:



The screenshot shows the Barracuda Email Security Gateway web interface. The top navigation bar includes tabs for BASIC, BLOCK/ACCEPT, USERS, DOMAINS, and ADVANCED. The 'Spam Checking' tab is selected. Below the navigation bar, there are links for Dashboard, Message Log, Spam Checking, Virus Checking, Quarantine, IP Configuration, Administration, Outbound, and Outbound Quarantine. The main content area is titled 'SPAM SCORING LIMITS' and contains three sections: Block, Quarantine, and Tag. Each section has a 'Disable' checkbox, a slider, and a numeric input field. The 'Block' section has a recommended score of 5. The 'Quarantine' section has a recommended score of 'Disable'. The 'Tag' section has a recommended score of 3.5. A 'Save' button is visible at the bottom right of the configuration area.

Conversely, most things that correspond to "action" type buttons in the web interface cannot be configured by the APIs. For example, from the **BASIC > Administration** page, you can click a button to take the system offline, to shut it down, or to clear the message log, but you cannot execute these "actions" via the APIs. An exception to this is the **Reload** feature/button – there is an API to re-apply the system configuration.



The screenshot shows the 'SYSTEM MANAGEMENT' section of the web interface. It contains several buttons: Clear Statistics, Clear Inbound Statistics, Clear Outbound Statistics, Clear Message Log, Shutdown, Restart, Offline, Reload, and Retry.

Understanding Variables in the Configuration

The examples in this guide demonstrate getting and setting some of the variables in the configuration database. Some examples use variable names in the method calls, while other examples use explicit values, just to demonstrate both ways of making API calls.

Make sure not to use an editor that may add special characters. Also make sure to use single quotes to surround literal values in your calls, and use double quotes to surround variables.

```
my $url      = "http://$cuda_ip:8000/cgi-mod/api.cgi?password=help";
my $result = $xmlrpc->call('config.set', {
    type => 'domain',
    path => 'barracuda.com',
    mta_relay_advanced_host => '1.3.3.7'
});
```

Two of the methods in the **General APIs** section, [config.varlist](#) and [config.var_attr](#), are utilities that provide information on scope and attributes of configuration variables to help you understand how to access and use them. Calling these methods prior to using the other APIs will provide a good reference of the configuration variables.

Secured Access to the APIs

Access to these APIs are limited to IP addresses on a trusted IP address list configured on the **BASIC > Administration** page in the **Allowed API IP/Range** section of the Barracuda Email Security Gateway web interface. Make sure to enter the IP address(es) from which you'll access the APIs in this section of the web interface as the first thing you do. Attempts to call these APIs from any IP address that is not on the list will be denied. All calls to the APIs require the use of the API password, which is set on the same page and section in the web interface.

Integration with External Systems and Services - Security Considerations

The Barracuda Email Security Gateway integrates with other systems and services in your environment, like your LDAP server and mail servers. Barracuda Networks recommends creating separate service accounts for these integration points, rather than personal accounts, and then using the principle of least privilege. This integration strategy is part of an overall security policy. For more information, see [Security for Integrating with Other Systems - Best Practices](#).

Barracuda Networks also strongly recommends sending API calls over HTTPS to protect passwords and other data.

XML-RPC Model

In the APIs, action parameters are received as XML strings that comply with the XML-RPC specification, which can be viewed here: <http://www.XMLrpc.com/spec.md>. This requires that requests for all actions be in the form of an HTTP POST request. All actions are rolled into one CGI script (for example: api.cgi) and map to an XML-RPC method, and the parameters are those needed for the action to complete.

For example, the **get** action maps to the config.get XML-RPC method and all the parameters needed

for the **get** will be sent in the XML body. The Perl module XML::RPC (note that this is not a part of the standard Perl distribution) will be used by api.cgi to retrieve the requested method and parameters. Once this is done, the action is performed and the response is sent back to the client. When an error is detected, a response that complies with the fault response of the XML-RPC specification is sent (see examples below). This response contains both a fault code and a meaningful fault string. See [Error Codes](#) for a list and explanation of fault codes.

The XML-RPC Request and Response

The XML script is called from a Perl script or other scripting language. Each API takes its own set of parameters which are submitted in the XML body of the request. Examples of the XML output follow the request example below, both for a successful request as well as for a request that returns an error. The single-value request / response involves a single variable value. Responses that contain multiple values will send the values back as an XML-RPC array. The response from the scripts is in the form of XML per the examples shown in this guide.

To make the request, use the base URL of your Barracuda Email Security Gateway that you use for connecting to the web interface and append the script name you wish to use. For example, if your script is called 'api.cgi', your URL might look something like this:

http://barracuda.mydomain.com:8000/cgi-mod/api.cgi

Parameters used to build the request typically include some or all of the following:

- **variable** – A required parameter that tells the API which variable to return from the configuration. For example, the configuration variable 'scana_block_level' represents the global Spam Scoring Limit block level as set on the BASIC > Spam Checking page in the web interface. To get or set this variable's value, you'd put 'scana_block_level' in the XML request body specified as a variable:

```
<name>variable</name>
<value>
  <string><![CDATA[scana_block_level]]></string>
</value>
```

- **password** – A required parameter which the API uses to authenticate access to a page and which is set by the administrator on the BASIC > Administration page in the API Password field. See the contents of 'my \$url' in the Single Value Request / Response example below, which uses a password of '1234'.
- **type** – A parameter that specifies the class/scope of a variable. The "scope" of a variable would be one of either global (for global settings), domain (for per-domain settings) or user (for per-user settings).

If the variable is a "tied variable", however, one or more other variables are related to it, so multiple variables will be specified in the XML request. For example, on the **BLOCK/ACCEPT > IP Reputation** page, a custom RBL domain name or IP address is associated with, or "tied to" an "action" of Block, Quarantine or Tag. The variable names to set, which you'll see in the configuration file, are `mta_rbl_custom_name` and `mta_rbl_custom_action` respectively. In this case, the "type" would be `'mta_rbl_custom_name'`.

- **path** - A parameter that typically corresponds to `scope_data` which refers to the particular instance of the object. For variables with global scope, the path is an empty string because there can be only one instance of global and it is the "starting point" in the same manner, for example, as the root (/) directory in Unix. So all variable and objects under global scope have type as `'global'` and path as an empty string.

When setting the value of a variable or variables that have a type of `'domain'`, the path would be expressed as the domain name. When working with tied variables such as `'httpd_acl_ip_config_address'` which relates to a value of `'httpd_acl_ip_config_netmask'`, for example, the path would be expressed as the actual IP address corresponding to `'httpd_acl_ip_config_address'`, as shown in this example:

To get the value of `httpd_acl_ip_config_netmask` corresponding to the `httpd_acl_ip_config_address` of `192.168.1.1`, the arguments would be:

```
type:      httpd_acl_ip_config_address
path:      192.168.1.1
variable:  httpd_acl_ip_config_netmask
```

Single Value Request / Response

To determine the **Spam Scoring Limit** global Block level (set in the web interface on the **BASIC > Spam Checking** page) for the Barracuda Email Security Gateway, use the `config.get` method to retrieve the current Block value as shown in this example.

To set the value of the global Block level, call the `config.set` method and set the variable `scana_block_level` to the desired value. Both calls deal with a single value. In the configuration, you'll see this entry for the global **Spam Scoring Limit** Block level, indicating that the current setting is '9' on the scale from 0-10:

```
# Default Block Level
scana_block_level = 9
```

Example: Perl

The config.get request would look something like this as called from a Perl script. The additional examples in further sections of this guide will only show the call from a Perl script.

```
#!/usr/bin/perl
use strict;
use LWP::UserAgent;
use HTTPS::Request::Common;
# IP Address of your Barracuda
my $cuda_ip = '192.168.126.98';
my $url      = "http://$cuda_ip:8000/cgi-mod/api.cgi?password=help" ;
my $ua = new LWP::UserAgent;
my $req = new HTTP::Request 'POST', $url;
my $xml = qq|
Here's the XML:
<?xml version="1.0" encoding="UTF8"?>
<methodCall>
  <methodName>config.get</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>variable</name>
            <value><string><![CDATA[scana_block_level]]></string>
          </value>
          </member>
          <member>
            <name>type</name>
            <value><string><![CDATA[global]]></string>
          </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
|;
# setup transport object with request object
$req->content_type('text/xml');
$req->content($xml);
# send the request over transport object
my $res = $ua->request($req);
# show the response from the Barracuda
```

```
print $res->as_string;  
# END
```

The request is an HTTP POST to the `/cgi-mod/api.cgi`. The POST data is an XML body that contains the **request** method `config.get` inside the `<methodName>` tag. The requested method is `config.get` since we are trying to retrieve the global Block level.

Note that the mandatory parameters needed for completing this action, "variable" (name of the configuration variable) and "password", are contained inside the `<struct>` tag. Each parameter is identified by the name (`<name>` tag) and the value (`<value>` tag). Possible types for each parameter are restricted by the types listed in the [XML-RPC specification](#).

This example includes only the mandatory parameters. Optional parameters can be added to the XML body using the format mentioned and will be processed accordingly. Sample output for the request would look something like this:

```
HTTP/1.1 200 OK  
Connection: close  
Date: Thu, 24 Jun 2010 18:41:47 GMT  
Server: BarracudaHTTP 2.0  
Content-Type: text/xml; charset=UTF-8  
Client-Date: Thu, 24 Jun 2010 18:42:08 GMT  
Client-Peer: 192.168.126.98:8000  
Client-Response-Num: 1  
Client-Transfer-Encoding: chunked
```

Here's the XML:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<methodResponse>  
  <methodName>config.get</methodName>  
  <params>  
    <param>  
      <value>  
        <i4>7</i4>  
      </value>  
    </param>  
  </params>  
</methodResponse>
```

All responses will contain the 200 OK success status code. Content-type of the response will be text/XML. The actual response, i.e. the value of the requested configuration variable, will be sent inside the `<value>` tag.

Multi-Value Response

Responses that contain multiple values will send the values back as an XML-RPC array. The example below is a request for a list of domains configured as Accepted Email Recipient Domain(s) on the Barracuda Email Security Gateway, which can be set from the **BASIC > IP Configuration** page in the web interface and which are stored in the configuration database in the domain variable.

The response may include multiple values, returned as an array inside the <array> tag. The format of the XML response body looks like this, returning three (domain name) values:

```
OK <?XML version="1.0" encoding="UTF8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>domain</name>
            <value>
              <array>
                <data>
                  <value>
                    <string>domain1.com</string>
                  </value>
                  <value>
                    <string>domain2.com</string>
                  </value>
                  <value>
                    <string> domain3.com</string>
                  </value>
                </data>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

Error responses use the XML-RPC faultCode and faultString formats. The error code will be the value of the faultCode member and the error string will be the valueError Response of the faultString member. See [Error Codes](#) for a list of faultCodes and descriptions of possible errors. Here's an example of an error response, showing the XML:


```
OK <?XML version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><i4>500</4></value>
        </member>
        <member>
          <name>faultString</name>
          <value>
            <string>No such variable in configuration</string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Example: PHP

This example calls the user.create API to create a new user account, which is covered in the **APIs for the Barracuda Email Security Gateway** section of this guide. The library used for this example can be found on the following SourceForge page: <http://sourceforge.net/projects/phpxmlrpc/>

In the code the library is included as a file. Make sure this file is readable from within your environment.

```
<?php
include("xmlrpc.inc");
$y = new xmlrpcval(
  array(
    "user" => new xmlrpcval("newuser@domain.com", "string")
  ), "struct");
$m = new xmlrpcmsg('user.create');
$m->addParam($y);
$c = new xmlrpc_client("/cgi-mod/api.cgi?password=[APIPassword]",
"[BarracudaIP]", [BarracudaPort]);
$r = $c->send($m);
if (!$r->faultCode()) {
  $v = $r->value();
  print $r->serialize();
} else {
```

```
print "Fault <BR>";
print "Code: " . htmlentities($r->faultCode()) . "<BR>" .
      "Reason: '" . htmlentities($r->faultString()) . "'<BR>";
}
?>
```

Example: Java

This example calls the user.create API to create a new user account, which is covered in the APIs for the Barracuda Email Security Gateway section of this guide. In the example, a key value pair is created using a standard Map class and added into a Vector list.

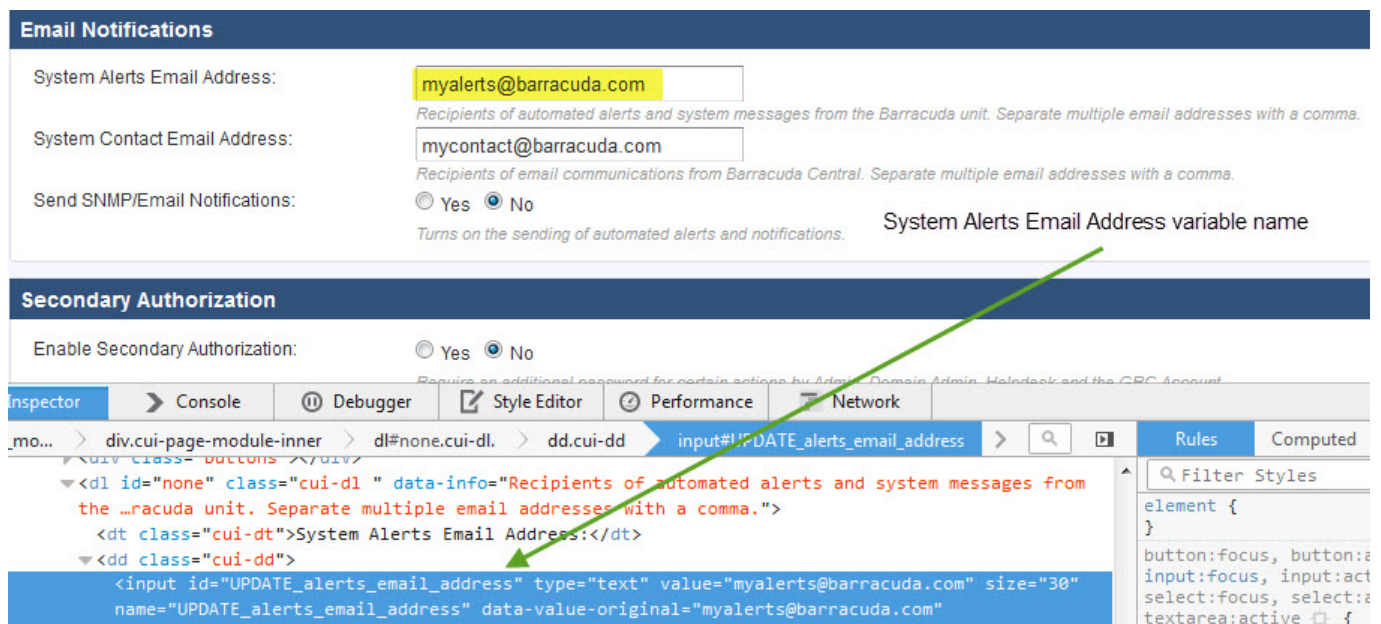
Required: Apache WS XML-RPC: <http://ws.apache.org/xmlrpc/>

```
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
import java.net.URL;
import java.util.Hashtable;
import java.util.Map;
import java.util.Vector;
public class BarracudaAPI {
    public static void main(String[] argv) {
        try {
            XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
            config.setServerURL(new
URL("https://[BarracudaIP]:[BarracudaPort]/cgi-mod/api.cgi?password=[APIPassw
ord]"));
            XmlRpcClient client = new XmlRpcClient();
            client.setConfig(config);
            // Create key value pair
            Map keyVals = new Hashtable();
            keyVals.put("user", "newuser@domain.com");
            // Start building the parameter list
            Vector params = new Vector();
            // Add key parameter
            params.add( keyVals );
            Object result = client.execute("user.create", params);
            System.out.println(result);
        } catch( Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

To determine the name of the variable you want to configure, log into the Barracuda Email Security

Gateway web interface as **admin** . On the page where you configure the setting, highlight the value field, right click and select Inspect Element. The *input_id* typically contains the name of the configuration variable. See the blue highlight in the figure below: the part of the *input_id* after UPDATE_ is the variable name. In this case, it is `alerts_email_address`.

How to Access Variables in the Configuration



Email Notifications

System Alerts Email Address:

Recipients of automated alerts and system messages from the Barracuda unit. Separate multiple email addresses with a comma.

System Contact Email Address:

Recipients of email communications from Barracuda Central. Separate multiple email addresses with a comma.

Send SNMP/Email Notifications: ☐ Yes ☒ No

Turns on the sending of automated alerts and notifications.

System Alerts Email Address variable name

Secondary Authorization

Enable Secondary Authorization: ☐ Yes ☒ No

Requires an additional password for certain actions by Admin, Domain Admin, Helpdesk and the GRC Account.

Inspector > Console > Debugger > Style Editor > Performance > Network

Rules > Computed

Filter Styles

element {

button:focus, button:active {

input:focus, input:active {

select:focus, select:active {

textarea:active {

input#UPDATE_alerts_email_address

<input id="UPDATE_alerts_email_address" type="text" value="myalerts@barracuda.com" size="30" name="UPDATE_alerts_email_address" data-value-original="myalerts@barracuda.com">

Figures

1. SpamScoring70BESG.png
2. SysMgmt70.png
3. SysAlertsVarName.png

© Barracuda Networks Inc., 2024 The information contained within this document is confidential and proprietary to Barracuda Networks Inc. No portion of this document may be copied, distributed, publicized or used for other than internal documentary purposes without the written consent of an official representative of Barracuda Networks Inc. All specifications are subject to change without notice. Barracuda Networks Inc. assumes no responsibility for any inaccuracies in this document. Barracuda Networks Inc. reserves the right to change, modify, transfer, or otherwise revise this publication without notice.