TRUSTWAVE SEG

SEG API Handler Listing 8.2

Table of Contents

About This Document

This document provides a formatted listing of the API commands as present in SEG 8.2.3.

8.2.3 introduces calls for Azure Information Protection status.

Changes from 8.1 to 8.2 include:

- DKIM generation key length option
- Import configuration DKIM password

For more information about the API and some examples of usage, see Trustwave Knowledge Base article Q20867.

Caution: API commands may change with product version. You should confirm the available functions from your installed version of SEG using the API List call described below. Calling applications should always confirm the API version before making other calls.

General notes

- In the URLs below, ^ indicates a path within the SEG API website. From the SEG Array Manager you can access this website at `https://localhost:19006/seg/api/`
- Variable substitutions within a path are shown as regular expressions.
- Names in `< >` indicate the type of item expected.
    - `(?<serverid>[0-9]+)` indicates the expected value is an integer server ID.
    - `(?<domain>.+)` indicates the expected value is a text domain name.
    - For example, to get Receiver details for node 1: `https://localhost:19006/seg/api/console/overview/1/receiver/`
    - To find the valid values, take the output of another call or see user interfaces.
- Query Parameters are sent as a http querystring. The parameter names are case sensitive.
- Variable data in querystrings MUST be URL Encoded. For example, a semi-colon ; must be represented as `%3B`
- POST data is generally a JSON formatted body.
- Times (integers) are Unix epoch timestamps. The minimum value is 0 and the maximum value is 2147483647. For testing, you can convert times using many free online services. For production, cast the date/time to this format programmatically.
- The parameter processMessageKey is a JSON object with the following items (all required): blockNumber, edition, folderId, messageName, recipient, serverId, timeLogged. These values can be obtained from the result of a Find Message call.

1 API List

This command allows you to get the current version of API commands from the server. Use Firefox or Postman to view the information in a clearly printed format.

Get API list

Method: GET
URL:`^debug/handlers/`

Example: `https://localhost:19006/seg/api/debug/handlers/`

2 ACLs

These commands allow you to find Windows user information for use in restricting Console access.

Get list of domains

Method: GET
URL:`^acl/domains/$`

Get list of groups

Method: GET
URL:`^acl/domains/(?<domain>.+)/groups/$`

Get list of users

Method: GET
URL:`^acl/domains/(?<domain>.+)/users/$`

Get sid from user account name

Method: GET
URL:`^acl/sid/(?<user>.+)/$`

Convert sids to users

- *Post data fields:* array of strings containing usernames

Method: POST
URL:`^acl/users/$`

3 Connectors

These commands allow you to work with SEG Connectors.

Get list of connectors

Method: GET
URL:`^connectors/$`

Update an existing connector

- *Put data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapVersion(str), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str) memberFilter(str), searchMembers(str), memberAttribute(str), addressFilter(str), mailAttribute(str), match(str), value(str), searchRoot(str)

Method: PUT
URL:`^connectors/$`

Create a new connector

- *Post data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapVersion(str), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str) memberFilter(str), searchMembers(str), memberAttribute(str), addressFilter(str), mailAttribute(str), match(str), value(str), searchRoot(str)

Method: POST
URL:`^connectors/$`

Get connector information

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/$`

Delete existing connector

Method: DELETE
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/$`

Get containers of a domain

- *Query parameters:* domain (str, dn - empty for all)

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/containers/$`

Get list of groups for a connector that have been imported into the GroupManager

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/importedgroups/$`

Get list of search groups for a container

- *Query parameters:* searchRoot (str), recursive (bool), nameFilter (str), descriptionFilter (str)

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/searchgroups/$`

Get list of search roots

Method: GET
URL:`^connectors/(?<connectorguid>[0-9A-Fa-f]{8}[-]?([0-9A-Fa-f]{4}[-]?){3}[0-9A-Fa-f]{12})/searchroots/$`

Get list of AD Domains

Method: GET
URL:`^connectors/addomains/$`

Test connector configuration

- *Post data:* guid(str), type(AD: or LDAP:), startTime (time_t), refreshDelay(int), name(str), description(str),
- *AD specific post data:* adLogonUser(str), adLogonPassword(str), adGcHost(str),
- *LDAP specific post data:* ldapServer(str), ldapVersion(str), ldapLogonUser(str), ldapLogonPassword(str), ldapLogonDomain(str) useSsl(bool), useNtlm(bool), timeout(int), groupClasses(str), nameAttribute(str), descriptionAttribute(str), userClasses(str) memberFilter(str), searchMembers(str), memberAttribute(str), addressFilter(str), mailAttribute(str), match(str), value(str), searchRoot(str)

Method: POST
URL:`^connectors/test/$`

4 Console

These commands allow you to work with the Console settings, statistics, and security, as well as retrying or killing a message.

Get a bitmap of console security settings

Method: GET
URL:`^console/accessmap/`

Get list of current alerts.

- *Query parameters:* activeonly(bool)

  Method: GET
  URL:`^console/alerts/`

Get array statistics.

- *Query parameters:* fromtime(int64), totime(int64)

  Method: GET
  URL:`^console/array/stats/`

Get list of event logs.

- *Query parameters:* servers(string), sources(string), logtype(integer), eventtype(integer), starttime(int64), endtime(int64), allsources(bool).
  - *logtype:* Application(1), Security(2), System(4), UpdateManager(8), All(15).
  - *eventtype:* Information(1), Warning(2), Error(4), Success(8), Failure(16), All(31)
  - *servers:* Semicolon separated list of node id numbers (blank = Array Manager)
  - *sources:* Semicolon separated list of service names.
  - *allsources:* If sources is blank and allsources=1, returns data for all services
  - Semicolons and service names must be URL Encoded (semicolon = `%3B`)

  Method: GET
  URL:`^console/event/logs/`

Get list of event sources

Method: GET
URL:`^console/event/sources/`

Get list of hold queue details for server

Method: GET
URL:`^console/holdqueue/details/(?<serverid>[0-9]+)/`

Retry the hold queue for server.

- *Post data fields:* ruleName(string)

  Method: POST
  URL:`^console/holdqueue/retry/(?<serverid>[0-9]+)/`

Get whether it is an appliance

Method: GET
URL:`^console/isappliance/`

Get whether mailbatching is enabled

Method: GET
URL:`^console/mailbatchingenabled/`

Get McAfee update information for server

Method: GET
URL:`^console/mcafeeupdateinfo/(?<serverid>[0-9]+)/`

Get server overview

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/$`

Get engine details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/engine/`

Get receiver details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/receiver/`

Get sender details

Method: GET
URL:`^console/overview/(?<serverid>[0-9]+)/sender/`

Get product update information.

- *Query parameters:* feedguid(string), brefresh(bool)

Method: GET
URL:`^console/productupdateinfo/`

Get whether quarantine message count on Today page is enabled

Method: GET
URL:`^console/quarantinecountsenabled/`

Kill message currently queued/being delivered by Sender.

- *Post data fields:* messageName(string), bNotify(bool)

Method: POST
URL:`^console/sender/message/kill/(?<serverid>[0-9]+)/`

Kill messages currently queued/being delivered by Sender.

- *Post data fields:* sender(string, regular expression), recipient(string, regular expression), subject(string, regular expression), speCustomerId(int) – at least one of these is required; bNotify(bool, optional)

Method: POST
URL:`^console/sender/message/killex/(?<serverid>[0-9]+)/`

Get sender route details for server.

- *Query parameters:* routename(string, optional), speCustomerId(int, optional) - at least one of routename or speCustomerId is required.

  Method: GET
  URL:`^console/sender/route/details/(?<serverid>[0-9]+)/`

Get sender route path for server.

- *Query parameters:* routename(string)

  Method: GET
  URL:`^console/sender/route/path/(?<serverid>[0-9]+)/`

Retry a route now in Sender.

- *Post data fields:* routeName(string)

  Method: POST
  URL:`^console/sender/route/processnow/(?<serverid>[0-9]+)/`

Get sender routes list for server.

- *Query parameters:* speCustomerId(int, optional)

  Method: GET
  URL:`^console/sender/routes/(?<serverid>[0-9]+)/`

Get list of disks information for server

Method: GET
URL:`^console/server/disks/(?<serverid>[0-9]+)/`

5 DMARC

Return timeseries graph of dmarc reports

- *Query parameters:* domain (str), fromDate (date), endDate (date), flags (int), renderLegend (bool), chartWidth (int), chartHeight(int)

  Method: GET
  URL:`^dmarc/reports/byday/graph/$`

Get raw DMARC reports

- *Query parameters:* fromDate (date), endDate (date), ipAddr (str), smtpDomain (str), reporterDomain (str), flags (int)

  Method: GET
  URL:`^dmarc/reports/raw/$`

Return top DMARC domains graph

- *Query parameters:* fromDate (date), endDate (date), chartWidth (int), chartHeight(int), renderLegend (bool), logScale (false), flags (int)

  Method: GET
  URL:`^dmarc/reports/topdomains/$`

Return top DMARC reporters graph

- *Query parameters:* fromDate (date), endDate (date), chartWidth (int), chartHeight(int), renderLegend (bool), logScale (false), flags (int)

  Method: GET
  URL:`^dmarc/reports/topreporters/$`

Return list of domains that we have received reports for

Method: GET
URL:`^dmarc/smtpdomains/$`

6 File Transfer

Create new file transfer session

Method: POST
URL:`^filetransfer/$`

Read file data

- *Query parameters:* bytes(int), offset(int: optional)

  Method: GET
  URL:`^filetransfer/(?<context>[0-9]+)/$`

Destroy a file transfer session

Method: DELETE
URL:`^filetransfer/(?<context>[0-9]+)/$`

Write file data

Method: PATCH
URL:`^filetransfer/(?<context>[0-9]+)/$`

7 Geolite (IP to location resolution)

Resolve the ipaddress to its geo location

Method: GET
URL:`^geolite/(?<ipaddr>.*)/$`

8 Groups

Get list of groups

- *Query parameters:* toplevelonly(bool, optional)

Method: GET
URL:`^groups/$`

Get group details

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/$`

Update group from connector

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/connectorupdate/$`

Add membership delta information to group

- *Patch data fields:* add(array of strings), remove(array of strings)

Method: PATCH
URL:`^groups/(?<groupid>[0-9]+)/delta/$`

Find which groups matches a user

- *Post data fields:* userEmail(string)

Method: POST
URL:`^groups/(?<groupid>[0-9]+)/find/$`

Get sub-group members of a group

Method: GET
URL:`^groups/(?<groupid>[0-9]+)/groups/$`

Locate user in groups

- *Post data fields:* userEmail(string, may include wildcard)

  Method: POST
  URL:`^groups/(?<groupid>[0-9]+)/locate/$`

Get group members

  Method: GET
  URL:`^groups/(?<groupid>[0-9]+)/members/$`

Import a group from a connector

- *Post data fields:* connectorGuid(string), dn(string)

  Method: POST
  URL:`^groups/import/$`

Create a new user-maintained group

- *Post data fields:* friendlyName(string), description(string), guid(string, optional)

  Method: POST
  URL:`^groups/usermaintained/$`

Update group

- *Put data fields:* friendlyName(string), description(string), pruningDays(int), pruningCount(int)

  Method: PUT
  URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Duplicate group

  Method: POST
  URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Delete group

  Method: DELETE
  URL:`^groups/usermaintained/(?<groupid>[0-9]+)/$`

Bulk set group members

- *Put data fields:* array of strings

  Method: PUT
  URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Add member to a group

- *Post data fields:* memberType(int), memberGroupId(int), memberData(string), touchDate(int)
    - *GroupMemberType:* 0 - group, 1 - email, 2 - wildcard

Method: POST
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Remove member from a group

- *Delete data fields:* memberType(int), memberGroupdId(int), memberData(string)
    - *GroupMemberType:* 0 - group, 1 - email, 2 - wildcard

Method: DELETE
URL:`^groups/usermaintained/(?<groupid>[0-9]+)/members/$`

Get group USN

- *Query parameters:* fqn(string)

Method: GET
URL:`^groups/usn/`

9 IP Groups

Get list of IP groups

Method: GET
URL:`^ipgroups/$`

Create a new IP group

- *Post data fields:* friendlyName(string), description(string), guid(string, optional)

Method: POST
URL:`^ipgroups/$`

Get IP group details

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Update an IP group

- *Put data fields:* friendlyName(string, optional), description(string, optional)

Method: PUT
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Delete an IP group

Method: DELETE
URL:`^ipgroups/(?<groupid>[0-9]+)/$`

Get child groups of an IP group

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/childgroups/$`

Add membership delta information to an IP group

- *Patch data fields:* add(array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string)), remove(array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string))

Method: PATCH
URL:`^ipgroups/(?<groupid>[0-9]+)/delta/$`

Duplicate an IP group

Method: POST
URL:`^ipgroups/(?<groupid>[0-9]+)/duplicate/$`

Find which IP group members match an IP address

- *Post data fields:* ip(string)

Method: POST
URL:`^ipgroups/(?<groupid>[0-9]+)/find/$`

Update an IP group member

- *Put data fields:* memberType(GroupMemberType, optional), oldName(string), newName(string), description(string, optional)
  - *GroupMemberType:* 3 - IPCIDR, 4 - IPRange

Method: PUT
URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Add a member to an IP group

- *Post data fields:* To insert Group: memberType(GroupMemberType), memberGroupdId(int); To add IPCIDR or IPRange: memberType(GroupMemberType), ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string), touchDate(int64)
  - *GroupMemberType:* 0 - Group, 3 - IPCIDR, 4 - IPRange

Method: POST
URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Remove a member from an IP group

- *Delete data fields:* To remove Group: memberType(GroupMemberType), memberGroupdId(int); To remove IPCIDR or IPRange: memberType(GroupMemberType), ipAddressStart(string), cidr(int), ipAddressEnd(string)
    - *GroupMemberType:* 0 - Group, 3 - IPCIDR, 4 - IPRange

  Method: DELETE
  URL:`^ipgroups/(?<groupid>[0-9]+)/member/$`

Get IP group members

Method: GET
URL:`^ipgroups/(?<groupid>[0-9]+)/members/$`

Bulk set IP group members

- *Put data fields:* array of objects with fields: ipAddressStart(string), cidr(int), ipAddressEnd(string, required only when cidr is 0), name(string), description(string)

  Method: PUT
  URL:`^ipgroups/(?<groupid>[0-9]+)/members/$`

Get IP group USN

- *Query parameters:* fqn(string)

  Method: GET
  URL:`^ipgroups/usn/`

10 Node TLS Commands

Get the current certificate details for the server

Method: GET
URL:`^nodetls/certificate/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional)

  Method: POST
  URL:`^nodetls/createcertificatesigningrequest/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), alternativeNames(array of strings)

Method: POST
URL:`^nodetls/createcertificatesigningrequest2/(?<serverid>[0-9]+)/$`

Generate a certificate signing request for the server, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createcertificatesigningrequest3/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long)

  Method: POST
  URL:`^nodetls/createprivatekey/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server, allowing to input subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createprivatekey2/(?<serverid>[0-9]+)/$`

Generate a self-signed certificate and its public-private key pair for the server, allowing to input signature hash algorithm and subject alternative names

- *Post data fields:* keyLength(long), country(string), state(string: optional), locality(string), org(string), ou(string), commonName(string), fqdn(string: optional), days(long), signMethod(int: 0 - sha1, 1 - sha256, 2 - sha512), alternativeNames(array of strings)

  Method: POST
  URL:`^nodetls/createprivatekey3/(?<serverid>[0-9]+)/$`

Generate a PKCS#12 backup file from the certificate and the private key currently installed on the server

- *Post data fields:* password(string: optional)

  Method: POST
  URL:`^nodetls/exportpkcs12file/(?<serverid>[0-9]+)/$`

Get whether the server has a pending certificate signing key

Method: GET
URL:`^nodetls/haspendingcertsigningkey/(?<serverid>[0-9]+)/$`

Get whether the server has a private key

Method: GET
URL:`^nodetls/hasprivatekey/(?<serverid>[0-9]+)/$`

Import a signed certificate supplied by a CA for the server

- *Post data fields:* certificate(string), privatekey(string), passphrase(string: optional)

Method: POST
URL:`^nodetls/importcertificate/(?<serverid>[0-9]+)/$`

Import a PKCS#12 backup file for the server

- *Post data fields:* data(base64 string), password(string: optional)

Method: POST
URL:`^nodetls/importpkcs12file/(?<serverid>[0-9]+)/$`

11 Quarantine (message handling)

Get folder day info

- *Query parameters:* utcOffset (int - in mins)

Method: GET
URL:`^quarantine/(?<folderid>[0-9]+)/(?<retention>[0-9]+)/$`

Get bulk folder day info

- *Post parameters:* map of "folder": retentionPeriod

Method: POST
URL:`^quarantine/folders/dayinfo/$`

Get list of classifications

Method: GET
URL:`^quarantine/classifications/$`

Find message by specified parameters

- *Post data:* startTime(int, required), endTime(int, required), folderId(int), messageName(str), actionType(int), classification(int), fromUser(str), fromDomain(str), toOrFrom(bool), toUser(str),

toDomain(str), minSize(int), maxSize(int), subject(str), searchHistory(bool), forwards(bool), blockNumber(int64), searchBlankSubject(bool), direction(int)
- *Query Parameters:* maxRows(int, required) *(if not set, no rows are)*

Method: POST
URL:`^quarantine/findmessage/$`

Find Receiver rejected message by specified parameters

- *Post data:* startTime(int), endTime(int), fromUser(str, optional), fromDomain(str, optional), toUser(str, optional), toDomain(str, optional), fromIPStart(str, optional), fromIPEnd(str, optional), speCustomerId(int, optional), forwards(bool, optional)
- *Query Parameters:* maxRows(int, required) *(if not set, no rows are returned)*

Method: POST
URL:`^quarantine/findrejectedmessage/$`

Get list of folders

- *Query parameters:* checkAccess (bool)

Method: GET
URL:`^quarantine/folders/$`

Get files for folderid

- *Query parameters:* filter (str), startTime (time_t), endTime (time_t), blockNumber (uint64), forwards (bool), maxMsgs (int)

Method: GET
URL:`^quarantine/folders/(?<folderid>[0-9]+)/$`

Check access for specific folder

Method: GET
URL:`^quarantine/folders/(?<folderid>[0-9]+)/checkaccess/$`

Get message count for a published folder

- *Query parameters:* user(recipient for published inbound, sender for published outbound: string)

Method: GET
URL:`^quarantine/folders/(?<folderid>[0-9]+)/messagecount/$`

Set folder properties

- *Post data:* ruleFolderItem (RuleFolderItem)

Method: PUT
URL:`^quarantine/folders/properties/$`

Get system folders

Method: GET
URL:`^quarantine/folders/system/$`

Forward a message to spiderlabs as spam

- *Post data:* messages (array of ProcessMessageKey), isSpam (bool), spamReportNotificationFromAddress (str)

Method: POST
URL:`^quarantine/forwardspam/$`

Delete specified messages

- *Post data:* messages (array of ProcessMessageKey), username (str), deleteType (int)

Method: DELETE
URL:`^quarantine/message/$`

Forward copy of message

- *Post data:* messages (array of ProcessMessageKey), deleteMessage (bool), recipients (str)
- *Optional post data:* speForwardCopyInfo (replyTo (str), fromAddr (str), speCustomerId (int), msgDirection (int))

Method: POST
URL:`^quarantine/message/forwardcopy/$`

Pass through message

- *Post data:* messages (array of ProcessMessageKey), userName (str), spamReportNotificationFromAddress (str), flags (int, required)

Method: POST
URL:`^quarantine/message/passthrough/$`

Get message properties

- *Post data:* messageKey field with json object with fields: blockNumber, edition, folderId, messageName, recipient, serverId, timeLogged

Method: POST
URL:`^quarantine/message/properties/$`

Restore specified messages from trash

- *Post data:* messages (array of ProcessMessageKey)

Method: POST
URL:`^quarantine/message/restore/$`

Trash specified messages

- *Post data:* messages (array of ProcessMessageKey)

Method: POST
URL:`^quarantine/message/trash/$`

Unpack message

- *Post data:* message (ProcessMessageKey), useContext (bool)

Method: POST
URL:`^quarantine/messagedetails/$`

Close unpacked message context

Method: DELETE
URL:`^quarantine/messagedetails/(?<contextid>[0-9]+)/$`

Retrieve unpacked message component

- *Query parameters:* offset, bytes

Method: GET
URL:`^quarantine/messagedetails/(?<contextid>[0-9]+)/(?<component>.*)/$`

Convert component html/rtf/text into form safe to display

- *Post parameters:* bodyType, body

Method: POST
URL:`^quarantine/messagedetails/makesafehtml/$`

Get latest messages

- *Query parameters:* user, messageCount

Method: GET
URL:`^quarantine/messages/latest/$`

Get list of published folders

Method: GET
URL:`^quarantine/publishedfolders/$`

Get list of files for published folder

- *Query parameters:* user(recipient for published inbound, sender for published outbound: string), maxMsgs(messages limit: int)

  Method: GET
  URL:`^quarantine/publishedfolders/(?<folderid>[0-9]+)/$`

Register SSM url

- *Post data:* url

  Method: POST
  URL:`^quarantine/registerssm/$`

Unpack message in sender queue

- *Post data:* messageName (string), useContext (bool)

  Method: POST
  URL:`^quarantine/sendermessagedetails/(?<serverid>[0-9]+)/$`

Delete quarantine trashed messages permanently

Method: DELETE
URL:`^quarantine/trash/$`

12 Services

Get a list of the AMAX script files

Method: GET
URL:`^services/amaxscriptfiles/$`

Check if the product is licensed for BTM

Method: GET
URL:`^services/btmlicensed/$`

Get current config timestamp

Method: GET
URL:`^services/config/current/$`

Export the configuration

Method: GET
URL:`^services/config/export/$`

Import configuration

- *Query parameters:* merge(bool), replacedll(bool), dkimPassword(string)

  Method: PUT
  URL:`^services/config/import/$`

Reload configuration for all servers.

- *Post data fields:* startStoppedServices(bool), forceOutOfSchedule(false)

  Method: POST
  URL:`^services/config/reload/$`

Reload server configuration.

- *Post data fields:* startStoppedServices(bool), forceOutOfSchedule(false)

  Method: POST
  URL:`^services/config/reload/(?<serverid>[0-9]+)/$`

Get database connection string used by ArrayManager to connect to database

Method: GET
URL:`^services/connectionstring/$`

Get the current admin user

Method: GET
URL:`^services/currentadminuser/$`

Check if delivery server is reachable or not.

- *Post data fields:* host(string), port(int)

  Method: POST
  URL:`^services/deliveryserver/check/(?<serverid>[0-9]+)/$`

Export password protected DKIM Private Key.

- *Query parameters:* passphrase(string)

  Method: GET
  URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Import the DKIM key for a domain

- *Put data fields:* key(base64 string), password(base64 string)

Method: PUT
URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Delete the DKIM key for a domain

Method: DELETE
URL:`^services/dkimkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Lookup DNS record and check status of DKIM public key

Method: GET
URL:`^services/dkimkey/dnsrecord/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Generate the DKIM key of specified keylength for a domain and selector

Method: POST
URL:`^services/dkimkey/generate/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/(?<keylength>[0-9]+)/$`

Get DKIM Public Key

Method: GET
URL:`^services/dkimkey/pubkey/(?<domainname>((?!/).)+)/(?<selector>((?!/).)+)/$`

Get the delivery routes for a domain

- *Query parameters:* specustomerid (int default=0), messagedirection (int default=0)

Method: GET
URL:`^services/domainroutes/(?<domainname>((?!/).)+)/$`

Run file update now

Method: POST
URL:`^services/fileupdate/run/$`

Get the license for SEG

Method: GET
URL:`^services/license/$`

Set the license for SEG

- *Put data fields:* licensekey(string)

Method: PUT
URL:`^services/license/$`

Create the initial license for SEG

Method: PUT
URL:`^services/license/createinitial/$`

Create a temporary license for SEG

Method: PUT
URL:`^services/license/createtemporary/$`

Request a license key

- *Post data fields:* reselleremail(string), contactname(string), contactemail(string), contactphone(string), contactcompany(string), contactaddress(string), customerreference(string), comments(string)

Method: POST
URL:`^services/license/request/$`

Get license key seeds

Method: GET
URL:`^services/license/seeds/$`

Get a list of SEG Node Ids

Method: GET
URL:`^services/nodeids/$`

Test the Marshal RBL service

- *Post data fields:* customernumber(string), rblkey(string)

Method: POST
URL:`^services/rbl/test/$`

Validate a regex against the built in regex library

- *Post data fields:* regex(string), ignorecase(bool)

Method: POST
URL:`^services/regex/$`

Get Rule profile details

- *Query parameters:* speCustomerId(int, optional)

Method: GET
URL:`^services/ruleprofile/(?<serverid>[0-9]+)/$`

Run script

Method: POST
URL:`^services/script/$`

Get security descriptor

Method: GET
URL:`^services/securitydescriptor/(?<oid>.+)/$`

Get security rights

Method: GET
URL:`^services/securityrights/$`

Get list of servers

Method: GET
URL:`^services/servers/$`

Get server details

Method: GET
URL:`^services/servers/(?<serverid>[0-9]+)/$`

Remove server from the array

Method: DELETE
URL:`^services/servers/(?<serverid>[0-9]+)/$`

Get service status for specified service on server

Method: GET
URL:`^services/servers/(?<serverid>[0-9]+)/(?<serviceid>[0-9]+)/$`

Start or stop service for specified service on server

- *Put data fields:* control(string) = "start" or "stop"

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/(?<serviceid>[0-9]+)/$`

Set server description

- *Put data fields:* description(string)

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/description/$`

Set server location

- *Put data fields:* location(string)

Method: PUT
URL:`^services/servers/(?<serverid>[0-9]+)/location/$`

Rename server

- *Put data fields:* name(string)

  Method: PUT
  URL:`^services/servers/(?<serverid>[0-9]+)/name/$`

Get a list of the server ip addresses

Method: GET
URL:`^services/servers/ips/(?<serverid>[0-9]+)/$`

Get the number of processors availaable on a server

Method: GET
URL:`^services/servers/processors/(?<serverid>[0-9]+)/$`

Request server upgrade

Method: GET
URL:`^services/servers/upgrade/(?<serverid>[0-9]+)/$`

Get details of the server update attempts

Method: GET
URL:`^services/serverupdatedata/$`

Get a list of SpamCensor types for a SpamCensor script

Method: GET
URL:`^services/spamcensortypes/(?<filename>.+)/$`

Get a list of SpamCensor files

Method: GET
URL:`^services/spamconfigfiles/$`

Convert a Text Censor line

- *Post data fields:* line(string), score(number), style(number)

  Method: POST
  URL:`^services/textcensor/convert/$`

Get user count

Method: GET
URL:`^services/usercount/$`

Get list of virus scanners

Method: GET
URL:`^services/virusscanners/$`

Test that a command line scanner works properly

- *Post data fields:* command(object), data(base64 string)
- *Command object data fields:* name(string), exefile(string), params(string), trigger(string), notrigger(string), timeout(number), timeoutmb(number)

Method: POST
URL:`^services/virusscanners/commandtest/(?<serverid>[0-9]+)/$`

Test that a command line or DLL virus scanner works properly

- *Post data fields:* dll(string), options(string), data(base64 string)

Method: POST
URL:`^services/virusscanners/dlltest/(?<serverid>[0-9]+)/$`

Get McAfee DAT file information

Method: GET
URL:`^services/virusscanners/mcafee/datinfo/(?<serverid>[0-9]+)/$`

Update McAfee DAT files

Method: GET
URL:`^services/virusscanners/mcafee/datupdate/(?<serverid>[0-9]+)/$`

Get McAfee status

Method: GET
URL:`^services/virusscanners/mcafee/status/(?<serverid>[0-9]+)/$`

13 Spam Quarantine Manager Site

Get authentication mode

Method: GET
URL:`^spam/authenticationmode/$`

Set authentication mode

- *Put data fields:* authMode(int: 0 - None, 1 - Forms, 2 - NTLM), enableADIntegration(boolean)

Method: PUT
URL:`^spam/authenticationmode/$`

Get whether global blacklist is enabled

Method: GET
URL:`^spam/globalblacklist/$`

Enable or disable global blacklist

- *Put data fields:* enabled(boolean)

  Method: PUT
  URL:`^spam/globalblacklist/$`

Get whether global per user digest is enabled

Method: GET
URL:`^spam/globalperuserdigest/$`

Enable or disable global per user digest

- *Put data fields:* enabled(boolean)

  Method: PUT
  URL:`^spam/globalperuserdigest/$`

Get whether global whitelist is enabled

Method: GET
URL:`^spam/globalwhitelist/$`

Enable or disable global whitelist

- *Put data fields:* enabled(boolean)

  Method: PUT
  URL:`^spam/globalwhitelist/$`

Get subscribable digest of a user. If user is null, get all digests and their default subscription

- *Query parameters:* user(string: optional)

  Method: GET
  URL:`^spam/subscribabledigests/$`

Set subscribable digest of a user. If user is null, the default subscription for the digest

- *Put data fields:* user(string: optional), digestName(string), isSubscribed(boolean)

  Method: PUT
  URL:`^spam/subscribabledigests/$`

Get a user's information by alias

- *Query parameters:* alias(string)

Method: GET
URL:`^spam/user/$`

Create a new user

- *Post data fields:* userName(string), fullName(string), updateTime(int64), sendDigest(boolean), languageCode(string), chartsEnabled(boolean), isAdministrator(boolean), createPassword(boolean), customPassword(string)

Method: POST
URL:`^spam/user/$`

Delete a user by userid

- *Query parameters:* userid(int)

Method: DELETE
URL:`^spam/user/$`

Get a user's information

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/$`

Update a user's information

- *Put data fields:* password(string), fullName(string), lastLoggedOn(int64), updateTime(int64), sendDigest(boolean), userRole(int), chartsEnabled(boolean), languageCode(string), lastMessageViewed(int64), theme(string)

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/$`

Create a new user

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/$`

Delete a user and associated aliases

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/$`

Delete an associated alias from a user

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/$`

Request to associate an alias to a user

- *Post data fields:* verificationUrl(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/requestassociation/$`

Verify the association of an alias and a user

- *Post data fields:* verificationCode(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/alias/(?<alias>((?!/).)+)/verifyassociation/$`

Get the list of aliases set up for this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/aliases/$`

Authenticate a user

- *Post data fields:* password(string)

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/authenticate/$`

Delete an alias from blocked list

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/blockedlist/alias/(?<alias>((?!/).)+)/$`

Allow another user to manage this user

Method: POST
URL:`^spam/user/(?<user>((?!/).)+)/delegate/(?<delegate>((?!/).)+)/$`

Delete an delegate from a user

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/delegate/(?<delegate>((?!/).)+)/$`

Get list of users who are allowed to manage this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/delegates/$`

Get list of users who are managed by this user

Method: GET
URL:`^spam/user/(?<user>((?!/).)+)/delegators/$`

Tell Array Manager that a user has been authenticated using NTLM

Method: PUT
URL:`^spam/user/(?<user>((?!/).)+)/ntlm/$`

Change a user's password

- *Put data fields:* oldPassword(string), newPassword(string)

  Method: PUT
  URL:`^spam/user/(?<user>((?!/).)+)/password/$`

Search user published messages by specific parameters

- *Post data fields:* fromDate(int64), toDate(int64), folderId(int), searchText(string), maxMsgs(int)

  Method: POST
  URL:`^spam/user/(?<user>((?!/).)+)/searchmail/$`

Get statistics of a user

- *Query parameters:* startTime(int64), stopTime(int64)

  Method: GET
  URL:`^spam/user/(?<user>((?!/).)+)/statistics/$`

Get whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

  Method: GET
  URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Bulk set whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)
  - *Put data fields:* entries(array of strings)

  Method: PUT
  URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Delete whitelist or blacklist of a user

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

  Method: DELETE
  URL:`^spam/user/(?<user>((?!/).)+)/whitelist/$`

Add an alias to whitelist or blacklist

- *Query parameters:* isWhitelist(int: 1 - whitelist, 0 - blacklist)

  Method: POST
  URL:`^spam/user/(?<user>((?!/).)+)/whitelist/alias/(?<alias>((?!/).)+)/$`

Delete an alias from whitelist

Method: DELETE
URL:`^spam/user/(?<user>((?!/).)+)/whitelist/alias/(?<alias>((?!/).)+)/$`

Get users information by key index

- *Query parameters:* keyIndex(char)

Method: GET
URL:`^spam/users/$`

Delete all users

Method: DELETE
URL:`^spam/users/$`

Backup users

Method: POST
URL:`^spam/users/backup/$`

Find users by matching a pattern against username

- *Post data fields:* searchPattern(string)

Method: POST
URL:`^spam/users/find/$`

Get list of key indexes of users

Method: GET
URL:`^spam/users/keyindexes/$`

Restore users

Method: POST
URL:`^spam/users/restore/$`

14 TextCensor

Get a specific Text Censor script

- *Query parameters:* name(string)

Method: GET
URL:`^textcensor/script/$`

Create or update a specific Text Censor script

- *Post data field:* Text Censor 2 script object (format as returned by GET)

Method: POST
URL:`^textcensor/script/$`

Delete a specific Text Censor script

- *Query parameters:* name(string)

Method: DELETE
URL:`^textcensor/script/$`

List defined Text Censor scripts

Method: GET
URL:`^textcensor/scripts/$`

Create a new TextCensor tester

Method: POST
URL:`^textcensor/testers/$`

Delete a TextCensor tester

Method: DELETE
URL:`^textcensor/testers/(?<context>[0-9]+)/$`

Get result of scanned file

Method: GET
URL:`^textcensor/testers/(?<context>[0-9]+)/evaluation/$`

Dump evaluation of script

Method: GET
URL:`^textcensor/testers/(?<context>[0-9]+)/evaluation/dump/$`

Add an expression to a TextCensor tester

- *Post data fields:* name(string), weight(integer), match(integer), expression(string)
  - *match:* match-limit
  - *expression:* TC expression base64-encoded

Method: POST
URL:`^textcensor/testers/(?<context>[0-9]+)/expressions/$`

Scan an uploaded file

Method: PUT
URL:`^textcensor/testers/(?<context>[0-9]+)/scan/(?<filecontext>[0-9]+)/$`

15 Azure Information Protection

Test AIP client

Method: GET
URL:^ services/aip/test_client/(?<serverid>[0-9]+)/$

Test AIP connection

- *Query parameters:* tenantid(string), appprincipalid(string), base64key(string), extraneturl(string), intraneturl(string)

Method: GET
URL:^services/aip/test_connection/(?<serverid>[0-9]+)/$

Get AIP error message

Method: GET
URL:^ services/aip/error_message/(?<serverid>[0-9]+)/$

16 Version information

Get version information

Method: GET
URL:^version/$

Get configuration version

Method: GET
URL:^version/config/

Get product version

Method: GET
URL:^version/product/

Get RPC interface version

Method: GET
URL:^version/rpcinterface/

About Trustwave

Trustwave helps businesses fight cybercrime, protect data and reduce security risk. With cloud and managed security services, integrated technologies and a team of security experts, ethical hackers and researchers, Trustwave enables businesses to transform the way they manage their information security and compliance programs. More than three million businesses are enrolled in the Trustwave TrustKeeper® cloud platform, through which Trustwave delivers automated, efficient and cost-effective threat, vulnerability and compliance management. Trustwave is headquartered in Chicago, with customers in 96 countries. For more information about Trustwave, visit https://www.trustwave.com.