PASSWORD RBL

API GUIDE

API VERSION 2.20

Table of Contents

Summary 3
Recommendations 3
API Endpoints 4
  Production Endpoints 4
  Development Endpoints 4
Method: Query 5
  GET request syntax 5
  Parameter Listing 5
  Required Parameters 5
    Required Parameter :: hashvalue 5
  Optional Parameters 7
    Optional Parameter :: apitype 7
    Optional Parameter :: trackingid 8
    Optional Parameter :: blacklistid 8
    Optional Parameter :: cblonly 8
Method: CBL-Management 9
  GET request syntax 9
  Parameter Listing 9
  Required Parameters 9
    Required Parameter :: action 9
    Required Parameter :: blacklistid 11
    Parameter :: hashvalue 11
Error Code Listing 12
API Version History 15

Summary

Use of the service is provided via a RESTful API over secure HTTPS transport. Customers have a choice of two hashing algorithms that can be used to securely submit pre-hashed passwords to the API using the “hashvalue” parameter. This is true when sending blacklist queries to search for a match as well as when managing a custom blacklist’s entries. Submissions are then hashed again upon reception before being inserted/removed from a custom blacklist or matched to Password RBL’s blacklist or a custom blacklist when using the query method.

The API only allows HTTPS GET requests. Parameters are passed to the API in the URL string as annotated in the API method calls later in this guide.

Recommendations

When implementing the Password RBL API on your web site or application, it is important to take into consideration all possible scenarios during your software development. Password RBL provides the following recommendations when developing software to use the Password RBL API.

- Backups – Before changing any production code base, it is important to have good, working and tested backups.
- Connectivity – The Password RBL API is a hosted solution located across the Internet and is therefore outside your completed control. It is important to consider scenarios when your software cannot make a successful connection to the API due to any number of unforeseen circumstances (Internet congestion, routing problems, etc.).
- API Responses – You should consider how your software will behave if the API returns an error code, rather than a normally formatted positive or negative result. Also, if you’ve exceeded your daily quota of blacklist queries, the API will reject your connection and instead reply with a TCP Reset packet.
- Certificates - Do not “hard code” or “memorize” any certificates or cryptographic keys in use by the API. Password RBL regularly changes certificates/keys.

API Endpoints

The API is available via two production endpoints and a development API endpoint. It is important to understand when to use each endpoint as they have different connectivity requirements.

Production Endpoints

- api.passwordrbl.com
  - This endpoint hosts the Query API method call
  - This endpoint is firewalled to only allow access from current subscribing systems.
  - Only one query per HTTP connection is allowed (keepalives are disabled).
- webservice.passwordrbl.com
  - This endpoint hosts the Custom Blacklist management API
  - This endpoint is firewalled to allow general access via HTTPS so connections can be made from management workstations.
  - Connections are throttled to only allow, on average, two connections per second, per source IP address. Keepalives are disabled.

Development Endpoints

- dev.passwordrbl.com
  - This is a development version of the 'api.passwordrbl.com' endpoint noted above.
  - It has the same connection restrictions as the production Query API endpoint
  - The blacklist available at this endpoint only has a few entries to use for testing code
  - This endpoint is available for free to customers and potential customers so that they can develop their Password RBL API implementation prior to beginning their subscription.
  - Contact your account representative or use the Contact form on the Password RBL website to arrange access and obtain the current DEV API documentation.

Method: Query

GET request syntax

https://api.passwordrbl.com/query.php?[*required_param*]&[*optional_params*]

Parameter Listing

Parameter	Required	Format / Value	Default
hashvalue	Yes	40 or 64 hex characters [0-9,a-f]	n/a
trackingid	No	32 hex characters [0-9,a-f]	n/a
blacklistid	No	32 hex characters [0-9,a-f]	n/a
cblonly	No	true | false	false
apitype	No	string | xml | json	string

Required Parameters

Required Parameter :: hashvalue

This is the only required parameter and is a salted and pre-hashed representation of the password submitted by your customer to your server. There are two industry-standard hashing algorithms to choose from, PBKDF2 or SHA256. It is not necessary to identify which algorithm was chosen when submitting queries to the API as each produce a different length value. PBKDF2 is preferred due to its inherent strength against brute force password-cracking attacks, so much so that it effectively makes it infeasible for anyone to reverse ("crack") the hashed value back to the original plaintext. SHA256 is provided for compatibility with systems that cannot perform the PBKDF2 algorithm.

Salt Value

It is important to note that both algorithms utilize a SALT value (defined below). The SALT value below is the salt value you must use. Do not choose your own SALT value or choose a random or changing SALT value. If you do not use this specific SALT value, then every submission to the API will result in a not-listed response.

```
SALT = "fe21a0daadda8301bf69a452963a2747a6c8aab4c016d9506a9af46b5f73a9ca"
```

PBKDF2

This is the recommended algorithm. This algorithm takes a password and SALT value as input and then performs many rounds of iterative hashing using a the SHA1 cryptographic hashing algorithm. All parameters of the PBKDF2 algorithm, except the password, have been pre-chosen and must match the parameters below:

Hash function: SHA1
Password: <provided by your customer>
SALT value: <see above>
Rounds: 30,000
Output Size: 20 bytes represented as 40 hex characters [0-9,a-f]

Example: `hashvalue = PBKDF2(sha1, Password, SALT, 30000, 20)`

SHA256

This algorithm is provided for compatibility. The output should be 64 hexadecimal characters and is obtained by appending the clear text password to the salt value (above) and passing the resulting string through the standard SHA256 algorithm. The result (hash) is then submitted to the Password RBL API via HTTPS GET request.

Example: `hashvalue = SHA256( concatenate(SALT, Password) )`

Optional Parameters

Optional Parameter :: apitype

This parameter designates what format you prefer to receive responses in. The default is String-format but XML and JSON formats are also available. Response values are listed in the following table:

Parameter	Default	Response Format
String	Yes	A single integer [0 | 1] to indicate existence in the Password RBL database or a negative value to indicate an error in the submission to the API.
XML	No	Content-type: text/xml <?xml version="1.0" encoding="utf-8" ?> <xmlresponse> <returnint> [null | 0 | 1] </returnint> <returnbool> [null | “true” | “false”] </returnbool> <error_code> [null | 0 | negative integer value] </error_code> <error_text> [text explanation of error] </error_text> </xmlresponse> NOTE: If the submission is of valid syntax, the values of the <returnint> and <returnbool> tags will have corresponding values to indicate existence of the submitted value in the Password RBL database. The values of the <error_code> and <error_text> tags will be null. If the submission is of invalid syntax, the <returnint> and <returnbool> tags will be null and the <error_code> and <error_text> tags will be filled with values that indicate the type of error.

JSON	No	Content-type: application/json
		{
		"jsonresponse":{
		"returnint": [null | 0 | 1] ,
		"returnbool": [null | “true” | “false”] ,
		"error_code": [null | 0 | negative integer value] ,
		"error_text": [null | text explanation of error]
		}
		}
		
		NOTE: If the submission is of valid syntax, the values of the “returnint” and “returnbool” tags will have corresponding values to indicate existence of the submitted value in the Password RBL database. The values of the “error_code” and “error_text” tags will be null.
		
		If the submission is of invalid syntax, the “returnint” and “returnbool” tags will be null and the “error_code” and “error_text” tags will be filled with values that indicate the type of error.

Optional Parameter :: trackingid

This is an optional parameter. The expected format is 32 hex characters. Queries to the Password RBL service are anonymous by default, but this prevents the service from providing customers with hit rate metrics. The customer can always perform the tracking of metrics on their own server/site. If you would like Password RBL to count queries to the API and how often each query results in a database match or not, you can supply a trackingID with each query. This allows for later reporting of these metrics using our metrics API or our online MyMetrics webpage.

Optional Parameter :: blacklistid

This is an optional parameter. The expected format is 32 hex characters. Queries that supply a blacklistID will perform the same iterative hashing of the supplied hashvalue, and will then search for a match in the identified custom blacklist. If a match is found, a positive response will be send back to the source. If a match is not found in the custom blacklist, then the API continues on to search for a match in the standard Password RBL password blacklist in the manner that would be taken if the query did not include the blacklistID parameter.

As of version 2.1, if a query includes a blacklistID and a trackingID, then metrics will be tracked in aggregate on the trackingID and metrics will also be tracked on the blacklistID, too. You can then use the MyMetrics page to receive a report for the trackingID and the blacklistID.

It is important to note, that if a blacklistID is specified but a trackingID is not, then metrics will not be tracked for the custom blacklist. A trackingID must be specified in order to track metrics on the custom blacklist.

Optional Parameter :: cblonly

This is an optional parameter. The expected format is either “true” or “false” and the default value is false. By default, queries that supply a blacklistID will search for a match in the specified custom blacklist and in the Password RBL maintained blacklist. Set this optional parameter to “true” and the API will only search for a match in the specified custom blacklist.

If the option is set to “false” or if this option is omitted, then the default behavior will occur.

If this option is set to “true” but a custom blacklist is not specified, an error is returned.

Method: CBL-Management

GET request syntax

https://webservice.passwordrbl.com/cbl-management.php?[*required_param*]&[*optional_params*]

Parameter Listing

Parameter	Required	Format / Value	Default
action	Yes	quota | count | add | delete | empty	n/a
blacklistid	Yes	32 hex characters [0-9,a-f]	n/a
hashvalue	Depends on action	40 or 64 hex characters [0-9,a-f]	n/a

Required Parameters

Required Parameter :: action

The action parameter is always required and can be one of the following: quota, count, add, delete, or empty. Each action type instructs the API to perform a specified action against the custom blacklist identified in the request. See below for a detailed explanation of each action type.

Quota

Submitting a query to the custom blacklist management method with the action type set to quota will return the maximum number of blacklist entries that are allowed for the specified custom blacklist. If you wish to increase the current quota assigned to your custom blacklist, simply use the contact form on the main website. Additional subscription fees may apply.

The quota action returns a positive integer that represents the maximum number of custom blacklist entries or a negative number to indicate error.

Count

Submitting a query to the custom blacklist management method with the action type set to count will return the current number of blacklist entries in your custom blacklist.

It is important to note that Password RBL supports multiple different hashing types with custom blacklists. Currently, PBKDF2 and SHA256 are supported, but others may be added in the future. It is only necessary to populate the custom blacklist with the hash type you will use. However, it is not detrimental to populate the custom blacklist with hashes of a type that you do not query. Therefore, Password RBL always recommends keeping the population of the hash types exactly the same. If you use the provided custom blacklist management tool, it populates both hash types, by default.

The count action will always return the maximum number of entries across all hash types.

For example, if you have 50 entries of type PBKDF2 and 100 entries of type SHA256, the count action will return 100.

Add

The add action adds the provided hashvalue to the custom blacklist.

The add action returns 1 if the add was successful, 0 if the add was unnecessary (syntactically correct but the entry was already in the blacklist), and a negative number to indicate error.

Delete

The delete action removes the provided hashvalue from the custom blacklist.

The delete action returns 1 if the removal was successful, 0 if the removal was unnecessary (syntactically correct but the entry was not found in the blacklist), and a negative number to indicate error.

Empty

The empty action removes all hash values of all types from the custom blacklist identified by the accompanied blacklistID in a single request.

The empty action returns the number of entries that were removed from the custom blacklist if the removal was successful, 0 if the removal was not successful, and a negative number to indicate error.

Required Parameter :: blacklistid

This parameter is always required. The expected format is 32 hex characters. This parameter identifies which custom blacklist in the Password RBL system is to be operated upon by the current request.

Parameter :: hashvalue

This parameter represents the hashed password that you will either add or remove from your custom blacklist. The hashvalue parameter is therefore required when the requested action is either add or delete, but is unused (and ignored if provided) when the action type is quota, count, or empty. The expected format is 40 or 64 hex characters, depending on the hash function that was utilized to produce the hash. PBKDF2 should output 40 hex characters whereas SHA256 produces 64 hex characters. The method for producing the hashvalue for use by custom blacklists is exactly the same as producing the hashvalues for use by the Query API call (see the section above for details).

Error Code Listing

Below is a listing of all the error codes that can be returned from the Password RBL API. This reference is especially helpful if you use the default string type of API response, since those return messages can only include the code and not the added explanation.

Error Code	Explanation
-410	Required parameter 'hashvalue' was not provided or was empty *The HTTPS GET request did not include the required URL parameter 'hashvalue' or the parameter was specified but had a null value.*
-411	Invalid format of HTTP parameter 'hashvalue' *The HTTPS GET request contains the required parameter 'hashvalue' but it was not the correct length or included non-hex characters.*
-412	Invalid format of HTTP parameter 'apitype' *The HTTPS GET request contained the optional parameter 'apitype' but the specified value was not one of the options specified by the API.*
-413	Invalid length of HTTP parameter 'trackingid' *The HTTPS GET request contained the optional parameter 'trackingid' but the specified value was not the correct length specified by the API.*
-414	Invalid format of HTTP parameter 'trackingid' *The HTTPS GET request contained the optional parameter 'trackingid' but the specified value contained non-hex characters.*
-415	Invalid length of HTTP parameter 'blacklistid' *The HTTPS GET request contained the optional parameter 'blacklistid' but the specified value was not the correct length specified by the API.*
-416	Invalid format of HTTP parameter 'blacklistid' *The HTTPS GET request contained the optional parameter 'blacklistid' but the specified value contained non-hex characters.*
-417	Invalid length of HTTP parameter 'cblonly' *The HTTPS GET request contained the optional parameter 'cblonly' but the specified value was not the correct length specified by the API.*
-418	Invalid format of HTTP parameter 'cblonly' *The HTTPS GET request contained the optional parameter 'cblonly' but the specified value was not equal to 'true' or 'false'*
-419	The parameter 'cblonly' was specified but 'blacklistid' was not. *The HTTPS GET request contained the optional parameter 'cblonly' but this option requires that a custom blacklist ID also be included in the same HTTPS GET request*

-421	The supplied ‘trackingid’ is not a valid ID but the format is valid *The HTTPS GET request contained the optional parameter ‘trackingid’ with valid syntax, but the specified value is not a valid Tracking ID.*
-451	Required parameter ‘action’ was not provided or was empty *The HTTPS GET request did not include the required parameter ‘action’ or the parameter was specified but had a null value.*
-452	Invalid format of HTTP parameter ‘action’ *The HTTPS GET request contained the required parameter ‘action’ but the specified value was not one of the options specified by the API*
-453	Required parameter ‘blacklistid’ was not provided or was empty *The HTTPS GET request did not include the required parameter ‘blacklistid’ or the parameter was specified but had a null value.*
-454	Invalid length of HTTP parameter ‘blacklistid’ *The HTTPS GET request contained the required parameter ‘blacklistid’ but the specified value was not the correct length specified by the API.*
-455	Invalid format of HTTP parameter ‘blacklistid’ *The HTTPS GET request contained the parameter ‘blacklistid’ but the specified value contained non-hex characters.*
-456	The supplied ‘blacklistid’ is not a valid ID but the format is valid *The HTTPS GET request contained the optional parameter ‘blacklistid’ with valid syntax, but the specified value is not a valid Blacklist ID.*
-457	There was an error executing the add command *This is a generic error that arises during the beginning processing of the XXX command, or if there is unexpected data returned from the backend database. If you receive this error, please contact us using the form on the website so we can look into this.*
-458	There was an error executing the add command *This error occurs if the hashvalue could not be added to the database. This would occur if communication was interrupted midstream or a configuration error. Please contact us if you receive this error.*
-460	There was an error executing the delete command *This error occurs if a database deletion was unsuccessful. If this error persists, please contact us.*
-461	There was an error executing the empty command for pbkdf2 entries. *This error database command to delete all pbkdf2 blacklist entries fails. If this error persists, please contact us.*
-462	There was an error executing the delete command for sha256 entries. *This error database command to delete all sha256 blacklist entries fails. If this error persists, please contact us.*

-501	Unable to connect to database *This is an internal error and occurs if the initial database connection fails.*
-502	Unable to connect to database *This is an internal error and occurs if the database connection fails during API processing.*

API Version History

Version	Notable Changes
Current Version	Added ‘cblonly’ parameter to control which blacklists to search when also using a Custom Blacklist.
2.10	Added metrics tracking on Custom Blacklists when a trackingID is also specified
2.00	Added Custom Blacklists and new “webservice” API Endpoint
1.60	Change default hashing algorithm to PBKDF2; SHA256 still supported for backwards compatibility
1.50	Change API parameter ‘sourceID’ to ‘trackingID’ to unify naming across offerings
1.41	Enhanced error messages with clear language
1.40	Enhanced verification and error reporting associated with ‘sourceID’ / ‘trackingID’ parameter
1.31	Enhanced JSON formatting output
1.30	Added JSON formatted API responses
1.20	Added ‘sourceID’ parameter in order to track metrics if customers decide to provide an optional tracking identifier to their queries.
1.10	Maintenance release; no notable customer-facing changes.
1.00	Original version