**SSL Labs API Documentation: v2.1**

**Last update:** 16 March 2015  
**Author:** Ivan Ristic [iristic@qualys.com](mailto:iristic@qualys.com)

This document explains the SSL Labs Assessment APIs, which can be used to test SSL servers available on the public Internet.

**Protocol Overview**

The protocol is based on HTTP and JSON. All invocations of the API should use the GET method and specify the parameters in the query string, as documented below. The results will be returned in the response body as a JSON payload. In essence, the client submits an assessment requests to the servers. If an acceptable report is already available, it's received straight away. Otherwise, the server will start a new assessment and the client should periodically check to see if the job is complete.

**Terms and Conditions**

SSL Labs APIs are provided free of charge, subject to our terms and conditions:<https://www.ssllabs.com/about/terms.html>. The spirit of the license is that the APIs are made available so that system operators can test their own infrastructure. Please read the actual terms and conditions, which are more involved and cover things such as integrating with open source projects, and so on. For example, it's important (for reasons of privacy, compliance, etc) for end users to understand that assessments are carried out by Qualys's servers, not locally.

Commercial use is generally not allowed, except with an explicit permission from Qualys. That said, we're usually happy to support good causes, even uses by commercial organizations that help improve the security of their customers. If you're a CA, CDN, hosting company, domain name registrar, we're happy for you to use our APIs (but you still have to get in touch with us before you begin).

**Protocol Calls**

This section documents the available protocol calls. The main API entry point is "<https://api.ssllabs.com/api/v2/>". If you want to check the API availability from a browser, invoke "<https://api.ssllabs.com/api/v2/info>". There is also an additional API entry point that can be used to test features that have not yet been deployed to production: "<https://api.dev.ssllabs.com/api/v2/>". You should expect that this second entry point is not consistently available. Further it offers only reduced assessment limits in comparison with the production version.

**Check SSL Labs availability**

This call should be used to check the availability of the SSL Labs servers, retrieve the engine and criteria version, and initialize the maximum number of concurrent assessments. Returns one [Info object](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#info) on success.

**API Call:** info

Parameters:

* None.

**Invoke assessment and check progress**

This call is used to initiate an assessment, or to retrieve the status of an assessment in progress or in the cache. It will return a single [Host object](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#host) on success. The Endpoint object embedded in the Host object will provide partial endpoint results.

**API Call:** analyze

Parameters:

* **host** - hostname; required.
* **publish** - set to "on" if assessment results should be published on the public results boards; optional, defaults to "off".
* **startNew** - if set to "on" then cached assessment results are ignored and a new assessment is started. However, if there's already an assessment in progress, its status is delivered instead. This parameter should be used only once to initiate a new assessment; further invocations should omit it to avoid causing an assessment loop.
* **fromCache** - always deliver cached assessment reports if available; optional, defaults to "off". This parameter is intended for API consumers that don't want to wait for assessment results. Can't be used at the same time as the startNew parameter.
* **maxAge** - maximum report age, in hours, if retrieving from cache (fromCache parameter set).
* **all** - by default this call results only summaries of individual endpoints. If this parameter is set to "on", full information will be returned. If set to "done", full information will be returned only if the assessment is complete (status is READY or ERROR).
* **ignoreMismatch** - set to "on" to proceed with assessments even when the server certificate doesn't match the assessment hostname. Set to off by default. Please note that this parameter is ignored if a cached report is returned.

Examples:

* /analyze?host=www.ssllabs.com
* /analyze?host=www.ssllabs.com&publish=on

**Retrieve detailed endpoint information**

This call is used to retrieve detailed endpoint information. It will return a single [Endpoint object](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#endpoint) on success. The object will contain complete assessment information. This API call does not initiate new assessments, even when a cached report is not found.

**API Call:** getEndpointData

Parameters:

* **host** - as above
* **s** - endpoint IP address
* **fromCache** - see above.

Example:

* /getEndpointData?host=www.ssllabs.com&s=173.203.82.166

**Retrieve known status codes**

This call will return one [StatusCodes instance](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#statuscodes).

**API Call:** getStatusCodes

Parameters:

* None.

**Protocol Usage**

When you want to obtain fresh test results for a particular host:

1. Invoke analyze with the startNew parameter to on. Set all to done.
2. The assessment is now in progress. Call analyze periodically (without the startNew parameter!) until the assessment is finished. You can tell by observing the Host.status field for either READY or ERROR values.
3. When there are multiple servers behind one hostname, they will be tested one at a time.
4. During the assessment, interim responses will contain only endpoint status, but not full information.
5. At the end of the assessment, the response will contain all available information; no further API calls will need to be made for that host.

When you're happy to receive cached information (e.g., in a browser add-on):

1. Invoke analyze with fromCache set to on and all set to done.
2. Set maxAge to control the maximum age of the cached report. If you don't set this parameter, your IP address will not be forwarded to the tested server.
3. If the information you requested is available in the cache, it will be returned straight away.
4. Otherwise, a new assessment will be started.
5. You can continue to call analyze periodically until the assessment is complete.

**Error Reporting**

When an API call is incorrectly invoked, it will cause an error response to be sent back. The response will include an array of error messages. For example:

{"errors":[{"field":"host","message":"qp.mandatory"}]}

The field value references the API parameter name that has an incorrect value. The message value will tell you what the issue is. It is also possible to receive errors without the field parameter set; such messages will usually refer to the request as a whole.

**Error Response Status Codes**

The following status codes are used:

* 400 - invocation error (e.g., invalid parameters)
* 429 - client request rate too high
* 500 - internal error
* 503 - the service is not available (e.g., down for maintenance)
* 529 - the service is overloaded

If you get 429, 503, 529, you should sleep for several minutes (e.g., 5, 15, 30 minutes, respectively) then try again. If you're writing an API client tool and get a 529 response, randomize the back-off time. If you get 500, it's best to give up.

**Access Rate and Rate Limiting**

Please note the following:

* Server assessments usually take at least 60 seconds. (They are intentionally slow, to avoid harming servers.) Thus, there is no need to poll for the results very often. In fact, polling too often slows down the service for everyone. It's best to use variable polling: 5 seconds until an assessment gets under way (status changes to IN\_PROGRESS), then 10 seconds until it completes.
* Keep down the number of concurrent assessments to a minimum. If you're not in a hurry, test only one hostname at a time.

We may limit your usage of the API, by enforcing a limit on concurrent assessments, and the overall number of assessments performed in a time period. If that happens, we will respond with 429 (Too Many Requests) to API calls that wish to initiate new assessments. Your ability to follow previously initiated assessments, or retrieve assessment results from the cache, will not be impacted. If you receive a 429 response, reduce the number of concurrent assessments.

If the server is overloaded (a condition that is not a result of the client's behaviour), the 529 status code will be used instead. This is not a situation we wish to be in. If you encounter it, take a break and come back after at least 30 minutes of sleep.

All successful API calls contain response headers X-Max-Assessments and X-Current-Assessments. They can be used to calculate how many new assessments can be submitted. It is recommended that clients update their internal state after each complete response.

**Protocol Evolution**

The API is versioned. New versions of the API will be introduced whenever incompatible changes need to be made to the protocol. When a new version becomes available, existing applications can continue to use the previous version for as long as it is supported.

To reduce version number inflation, new fields may be added to the results without a change in protocol version number.

**Response Objects**

The remainder of the document explains the structure of the returned objects. The following conventions are used:

* **field** - a simple field
* **object{}** - an object
* **array[]** - an array

**Host**

* **host** - assessment host, which can be a hostname or an IP address
* **port** - assessment port (e.g., 443)
* **protocol** - protocol (e.g., HTTP)
* **isPublic** - true if this assessment publicly available (listed on the SSL Labs assessment boards)
* **status** - assessment status; possible values: DNS, ERROR, IN\_PROGRESS, and READY.
* **statusMessage** - status message in English. When status is ERROR, this field will contain an error message.
* **startTime** - assessment starting time, in milliseconds since 1970
* **testTime** - assessment completion time, in milliseconds since 1970
* **engineVersion** - assessment engine version (e.g., "1.0.120")
* **criteriaVersion** - grading criteria version (e.g., "2009")
* **cacheExpiryTime** - when will the assessment results expire from the cache (typically set only for assessment with errors; otherwise the results stay in the cache for as long as there's sufficient room)
* **endpoints[]** - list of [Endpoint objects](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#endpoint)
* **certHostnames[]** - the list of certificate hostnames collected from the certificates seen during assessment. The hostnames may not be valid.

**Endpoint**

* **ipAddress** - endpoint IP address, in IPv4 or IPv6 format.
* **serverName** - server name retrieved via reverse DNS
* **statusMessage** - assessment status message
* **statusDetails** - code of the operation currently in progress
* **statusDetailsMessage** - description of the operation currently in progress
* **grade** - possible values: A+, A-, A-F, T (no trust) and M (certificate name mismatch)
* **hasWarnings** - if this endpoint has warnings that might affect the score (e.g., get A- instead of A).
* **isExceptional** - this flag will be raised when an exceptional configuration is encountered. The SSL Labs test will give such sites an A+.
* **progress** - assessment progress, which is a value from 0 to 100, and -1 if the assessment has not yet started
* **duration** - assessment duration, in milliseconds
* **eta** - estimated time, in seconds, until the completion of the assessment
* **delegation** - indicates domain name delegation with and without the www prefix
  + bit 0 (1) is set for non-prefixed access
  + bit 1 (2) is set for prefixed access
* **details** - this field contains an EndpointDetails object. It's not present by default, but can be enabled by using the "all" paramerer to the analyze API call.

**EndpointDetails**

* **hostStartTime** = endpoint assessment starting time, in milliseconds since 1970. This field is useful when test results are retrieved in several HTTP invocations. Then, you should check that the hostStartTime value matches the startTime value of the host.
* **key{}** - [key information](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#key)
* **cert{}** - [certificate information](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#cert)
* **chain{}** - [chain information](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#chain)
* **protocols[]** - supported [protocols](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#protocol)
* **suites{}** - supported [cipher suites](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#suites)
* **serverSignature** - Contents of the HTTP Server response header when known. This field could be absent for one of two reasons: 1) the HTTP request failed (check httpStatusCode) or 2) there was no Server response header returned.
* **prefixDelegation** - true if this endpoint is reachable via a hostname with the www prefix
* **nonPrefixDelegation** (moved here from the summary) - true if this endpoint is reachable via a hostname without the www prefix
* **vulnBeast** - true if the endpoint is vulnerable to the BEAST attack
* **renegSupport** - this is an integer value that describes the endpoint support for renegotiation:
  + bit 0 (1) is set if insecure client-initiated renegotiation is supported
  + bit 1 (2) is set if secure renegotiation is supported
  + bit 2 (4) is set if secure client-initiated renegotiation is supported
  + bit 3 (8) is set if the server requires secure renegotiation support
* **stsResponseHeader** - the contents of the Strict-Transport-Security (STS) response header, if seen
* **stsMaxAge** - the maxAge parameter extracted from the STS parameters; null if STS not seen, or -1 if the specified value is invalid (e.g., not a zero or a positive integer; the maximum value currently supported is 2,147,483,647)
* **stsSubdomains** - true if the includeSubDomains STS parameter is set; null if STS not seen
* **pkpResponseHeader** - the contents of the Public-Key-Pinning response header, if seen
* **sessionResumption** - this is an integer value that describes endpoint support for session resumption. The possible values are:
  + 0 - session resumption is not enabled and we're seeing empty session IDs
  + 1 - endpoint returns session IDs, but sessions are not resumed
  + 2 - session resumption is enabled
* **compressionMethods** - integer value that describes supported compression methods
  + bit 0 is set for DEFLATE
* **supportsNpn** - true if the server supports NPN
* **npnProtocols** - space separated list of supported protocols
* **sessionTickets** - indicates support for Session Tickets
  + bit 0 is set if session tickets are supported
  + bit 1 (not implemented) is set if the implementation is faulty
  + bit 2 is set if the server is intolerant to the extension
* **ocspStapling** - true if OCSP stapling is deployed on the server
* **sniRequired** - if SNI support is required to access the web site.
* **httpStatusCode** - status code of the final HTTP response seen. When submitting HTTP requests, redirections are followed, but only if they lead to the same hostname. If this field is not available, that means the HTTP request failed.
* **httpForwarding** - available on a server that responded with a redirection to some other hostname.
* **supportsRc4** - true if the server supports at least one RC4 suite.
* **forwardSecrecy** - indicates support for Forward Secrecy
  + bit 0 - set if at least one browser from our simulations negotiated a Forward Secrecy suite.
  + bit 1 - set based on Simulator results if FS is achieved with modern clients. For example, the server supports ECDHE suites, but not DHE.
  + bit 2 - set if all simulated clients achieve FS. In other words, this requires an ECDHE + DHE combination to be supported.
* **rc4WithModern** - true if RC4 is used with modern clients.
* **sims** - instance of [SimDetails](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#simdetails).
* **heartbleed** - true if the server is vulnerable to the Heartbleed attack.
* **heartbeat** - true if the server supports the Heartbeat extension.
* **openSslCcs** - results of the CVE-2014-0224 test:
  + -1 - test failed
  + 0 - unknown
  + 1 - not vulnerable
  + 2 - possibly vulnerable, but not exploitable
  + 3 - vulnerable and exploitable
* **poodleTls** - results of the POODLE TLS test:
  + -1 - test failed
  + 0 - unknown
  + 1 - not vulnerable
  + 2 - vulnerable
* **fallbackScsv** - true if the server supports TLS\_FALLBACK\_SCSV, false if it doesn't. This field will not be available if the server's support for TLS\_FALLBACK\_SCSV can't be tested because it supports only one protocol version (e.g., only TLS 1.2).

**Info**

* **version** - SSL Labs software version as a string (e.g., "1.11.14")
* **criteriaVersion** - rating criteria version as a string (e.g., "2009f")
* **maxAssessments** - the maximum number of concurrent assessments the client is allowed to initiate.
* **currentAssessments** - the number of ongoing assessments submitted by this client.
* **messages** - a list of messages (strings). Messages can be public (sent to everyone) and private (sent only to the invoking client). Private messages are prefixed with "[Private]".
* **clientMaxAssessments** - deprecated and scheduled for removal in the next release.

**Key**

* **size** - key size, e.g., 1024 or 2048 for RSA and DSA, or 256 bits for EC.
* **strength** - key size expressed in RSA bits.
* **alg** - key algorithm; possible values: RSA, DSA, and EC.
* **debianFlaw** - true if we suspect that the key was generated using a weak random number generator (detected via a blacklist database)
* **q** - 0 if key is insecure, null otherwise

**Cert**

* **subject** - certificate subject
* **commonNames[]** - common names extracted from the subject
* **altNames[]** - alternative names
* **notBefore** - timestamp before which the certificate is not valid
* **notAfter** - timestamp after which the certificate is not valid
* **issuerSubject** - issuer subject
* **sigAlg** - certificate signature algorithm
* **issuerLabel** - issuer name
* **revocationInfo** - a number that represents revocation information present in the certificate:
  + bit 0 (1) - CRL information available
  + bit 1 (2) - OCSP information available
* **crlURIs[]** - CRL URIs extracted from the certificate
* **ocspURIs[]** - OCSP URIs extracted from the certificate
* **revocationStatus** - a number that describes the revocation status of the certificate:
  + 0 - not checked
  + 1 - certificate revoked
  + 2 - certificate not revoked
  + 3 - revocation check error
* **sgc** - Server Gated Cryptography support; integer:
  + bit 1 (1) - Netscape SGC
  + bit 2 (2) - Microsoft SGC
* **validationType** - E for Extended Validation certificates; may be null if unable to determine
* **issues** - list of certificate issues, one bit per issue:
  + bit 0 (1) - no chain of trust
  + bit 1 (2) - not before
  + bit 2 (4) - not after
  + bit 3 (8) - hostname mismatch
  + bit 4 (16) - revoked
  + bit 5 (32) - bad common name
  + bit 6 (64) - self-signed
  + bit 7 (128) - blacklisted
  + bit 8 (256) - insecure signature

**Chain**

* **certs[]** - a list of [ChainCert objects](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#chaincert), representing the chain certificates in the order in which they were retrieved from the server
* **issues** - a number of flags that describe the chain and the problems it has:
  + bit 0 (1) - unused
  + bit 1 (2) - incomplete chain (set only when we were able to build a chain by adding missing intermediate certificates from external sources)
  + bit 2 (4) - chain contains unrelated or duplicate certificates (i.e., certificates that are not part of the same chain)
  + bit 3 (8) - the certificates form a chain (trusted or not), but the order is incorrect
  + bit 4 (16) - contains a self-signed root certificate (not set for self-signed leafs)
  + bit 5 (32) - the certificates form a chain (if we added external certificates, bit 1 will be set), but we could not validate it. If the leaf was trusted, that means that we built a different chain we trusted.

**ChainCert**

* **subject** - certificate subject
* **label** - certificate label (user-friendly name)
* **issuerSubject** - issuer subject
* **issuerLabel** - issuer label (user-friendly name)
* **issues** - a number of flags the describe the problems with this certificate:
  + bit 0 (1) - certificate not yet valid
  + bit 1 (2) - certificate expired
  + bit 2 (4) - weak key
  + bit 3 (8) - weak signature
  + bit 4 (16) - blacklisted
* **raw** - PEM-encoded certificate data

**Protocol**

* **id** - protocol version number, e.g. 0x0303 for TLS 1.2
* **name** - protocol name, i.e. SSL or TLS.
* **version** - protocol version, e.g. 1.2 (for TLS)
* **v2SuitesDisabled** - some servers have SSLv2 protocol enabled, but with all SSLv2 cipher suites disabled. In that case, this field is set to true.
* **q** - 0 if the protocol is insecure, null otherwise

**SimClient**

* **id** - unique client ID (integer)
* **name** - text.
* **platform** - text.
* **version** - text.
* **isReference** - true if the browser is considered representative of modern browsers, false otherwise. This flag does not correlate to client's capabilities, but is used by SSL Labs to determine if a particular configuration is effective. For example, to track Forward Secrecy support, we mark several representative browsers as "modern" and then test to see if they succeed in negotiating a FS suite. Just as an illustration, modern browsers are currently Chrome, Firefox (not ESR versions), IE/Win7, and Safari.

**SimDetails**

* **results[]** - instances of [Simulation](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#simulation).

**Simulation**

* **client** - instance of [SimClient](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#simclient).
* **errorCode** - zero if handshake was successful, 1 if it was not.
* **attempts** - always 1 with the current implementation.
* **protocolId** - Negotiated protocol ID.
* **suiteId** - Negotiated suite ID.

**Suites**

* **list[]** - list of [Suite objects](https://github.com/ssllabs/ssllabs-scan/blob/stable/ssllabs-api-docs.md#suite)
* **preference** - true if the server actively selects cipher suites; if null, we were not able to determine if the server has a preference

**Suite**

* **id** - suite RFC ID (e.g., 5)
* **name** - suite name (e.g., TLS\_RSA\_WITH\_RC4\_128\_SHA)
* **cipherStrength** - suite strength (e.g., 128)
* **dhStrength** - strength of DH params (e.g., 1024)
* **dhP** - DH params, p component
* **dhG** - DH params, g component
* **dhYs** - DH params, Ys component
* **ecdhBits** - ECDH bits
* **ecdhStrength** - ECDH RSA-equivalent strength
* **q** - 0 if the suite is insecure, null otherwise

**StatusCodes**

* **statusDetails** - a map containing all status details codes and the corresponding English translations. Please note that, once in use, the codes will not change, whereas the translations may change at any time.

**Changes**

**2.0 (3 March 2015)**

* First public release.

**2.1 (16 March 2015)**

* Added ignoreMismatch parameter to control if assessments proceed when server certificate does not match the assessment hostname.