SonicWall™ SSO API

Reference Guide

Contents

1

SSO Overview

This document specifies the external application program interface (API) for third party devices, software, or scripts to pass user login/logout notifications directly to the SonicWall firewall for SSO API. Its aim is to provide general specifications for the API, guidance for using it, and instructions for writing client-side software or scripts for it.

Topics:

Description

This SSO (Single Sign-On) API allows external third-party devices, software or scripts to pass user login/logout notifications directly to the SonicWall firewall for SSO.

The architecture of the API is designed to be simple, flexible and easy to use for scripting by customers, while still being robust and fully secure. It is based on the REST (REpresentational State Transfer), or RESTful, architecture. This is a loosely defined mechanism for creating, reading and updating web resources using HTTP methods (POST, GET, DELETE, and so forth). In the case of this API, those resources are the users logged into the firewall. The API supports the methods POST to add users (log them in) and DELETE to delete them (log them out), plus the OPTIONS request for clients to query information about the API.

Third-party clients that have been enabled to do so on the SonicWall firewall can use the API. It needs to send HTTPS requests to the API indicating user logins and user logouts. The SonicWall firewall can then apply configured, per-user policies for access rights, which may include user group memberships supplied by either the API or looked up via LDAP. Refer to Configuration for more information.

REST supports the request content being either XML or JSON. XML is the default, but JSON is often the preferred choice since it is simpler and more compact. In this API, the SonicWALL firewall supports both XML and JSON, so the you can choose which to use.

The information in this document forms a guide for using the API and for writing client-side software or scripts for it. Sample scripts that can be used as a reference are available on request.

Terms

This SSO API allows external third-party devices, software or scripts to pass user login/logout notifications directly to the SonicWall firewall for Single Sign On. The following terms are used within this document:

CSRF	Cross Site Request Forgery. The sending of malicious requests by malware that can generate and send such requests from the victim's own computer.
JSON	JavaScript Object Notation. A syntax for storing and exchanging data in a format very similar to JavaScript Objects that is both human and machine readable.
REST	Representational State Transfer Architecture. Architecture for simple and efficient stateless access to web resources using HTTP methods.
REST API or **RESTful API**	An API for clients to access or update resources on a web server using the REST architecture with XML or JSON data. The two terms are interchangeable.
SSO	Single sign-on. The ability of the SonicWall firewall to authenticate users sending traffic through it in a way that is transparent to those users; that is, the users don't need to go through any additional login process to identify themselves to the firewall.
XML	Extensible Markup Language. A syntax for storing and exchanging data in a format that is both human and machine readable.

Other Aspects

Other features of the SSO API include:

- Use with an External Authentication Page
- Backward Compatibility

Use with an External Authentication Page

One possible way to utilize this SSO API is to have the SonicWall firewall redirect unauthenticated HTTP/HTTPS connections to a customer-provided external login page. The latter would then initiate authentication of the sender using its own authentication system. When the users are authenticated and logged in, this SSO API can be used to notify the SonicWall firewall.

Note that the redirection for this can be configured on the SonicWall firewall. Refer to Configuration for more details.

Backward Compatibility

This document specifies Version 1 of the SSO API. Although the API may be changed, enhanced or extended in the future, such future versions of the API are required to retain backwards compatibility with existing clients. So long as a client conforms to the API as specified here, your current API should be able to send requests to any future version of the SonicWall firewall and have them operate as detailed here.

2

Requests and Responses

Topics:

Request Format

Messages passed over the SSO API are standard web HTTP requests. They are sent over HTTPS to the firewall's web server, forming a REST-conformant API as described here.

REST requires that the URI in a request identifies a *collection* of resources being read or added to or, in a request for an existing resource, that it identifies the specific *resource* that it is targeting. In this API, the *collection* is all the logged-in users and the *resource* is a specific user.

The URI begins with a path that identifies the web service and/or collection (SSO and users, respectively). For requests for an existing resource (user) that is followed by an identifier for the resource, that can be anything which uniquely identifies it. The initial request line in the HTTP header of an API request gives the method, its accompanying URI, and identifies a user by his or her IP address:

`POST api/sso/user HTTP/1.1`	To log in a user (with the IP address given in the content).
`DELETE api/sso/user/ip-address HTTP/1.1`	To log out the user at the given IP address.

In REST terms, the URI of a POST request specifies the collection (users) that it is adding to, and that of a DELETE request identifies the specific user that is being deleted.

NOTE: A POST should not identify the user in the URI because that user does not exist until the POST has been completed. The URI is being asked to identify a resource and logically can't do that until the resource exists.

REST also specifies that requests passing content should give the format of the content in the request's **Content-Type** header. If the content type is not specified in a header, the default is to expect content in XML. This API complies with this and accepts the following:

```
Content-Type: application/json
Content-Type: application/xml
```

In cases where the SonicWall firewall returns body content in its reply to the client (during the OPTIONS request, for example) it uses XML or JSON per the request's **Accept** header. If the header indicates that the client can accept both, or if the header is not present, it uses the same type as was used in the request. It defaults to XML if the request had no body content. A corresponding **Content-Type** header is included in the reply.

Content Format

The following specifies what the URI and body content of the request should hold for the supported operations. All requests support passing the data as either XML or JSON. The names and content of all the attributes are the same in both cases, just packaged differently.

User Login

Topics:

- Single Login
- Multiple Login

Single Login

A new login of a single user is indicated by:

```
POST api/sso/user HTTP/1.1
```

And the body content is:

JSON

```
{ "ip": "ip-address", "name": "user-name", "domain": "domain-name",
"type": "type-of-user", … }
```

XML

```
<user>
     <ip>ip-address</ip>
     <name>user-name</name>
     <domain>domain-name</domain>
     <type>type-of-user</type>
     …
</user>
```

Refer to User Attributes for a list of the user attributes than can be posted with this.

Multiple Login

A notification of login of multiple users can be sent in a single request, with the same URI:

```
POST api/sso/user HTTP/1.1
```

The body content is:

JSON

```
[
     { "ip": "ip-address-1", "name": "user-name", "domain": "dom-name",
     "type": "type-of-user", … },
     { "ip": "ip-address-2", "name": "user-name", "domain": "dom-name",
     "type": "type-of-user", … }
]
```

XML

```
    <users>
        <user>
            <ip>ip-address-1</ip>
            <name>user-name</name>
            <domain>domain-name</domain>
            <type>type-of-user</type>
        </user>
        <user>
            <ip>ip-address-2</ip>
            <name>user-name</name>
            <domain>domain-name</domain>
            <type>type-of-user</type>
        </user>
        ...
    </users>
```

The content of each user entry is the same as above for a single user post. Posting in this format while containing just one user entry is allowed.

User Logout

Topics:

- Single Logout
- Logout with an External IP Address
- Multiple Logout

Single Logout

A logout of a single user is indicated by:

```
    DELETE api/sso/user/ip-address HTTP/1.1
```

No body content is needed, but passing some information in the body is supported. The information passed is used only for logging purposes.

NOTE: The user is logged out whether or not any given name and/or domain match those of the logged in user.

If any body content is present (all fields are options), it should follows this format:

JSON

```
{ "name": "user-name", "domain": "domain-name", "reason": "text" }
```

XML

```
<user>
     <name>user-name</name>
     <domain>domain-name</name>
     <reason>text</reason>
</user>
```

Refer to User Attributes for a definition of these user attributes. Any attributes other than those shown above and as noted below are silently ignored.

Logout with an External IP Address

When passing data in the body of a DELETE request as above, the **ip-remote** attribute can be included to tell the firewall that the user IP address in the URI is an external or remote IP address that may need to be NAT translated. (Refer to User Attributes for more information.)

If you would rather not include content data in the DELETE request, or if cannot include content data, you can use the following string query on the URI instead:

```
DELETE api/sso/user/<ip-address>?ip-remote=true HTTP/1.1
```

If a DELETE request does include content data, the SonicWall firewall attempts to NAT-translate the given IP address under these conditions:

- **ip-remote** is present
- The value **true** is in either the content data or the URI

Multiple Logout

A notification of multiple user logouts can be sent in a single request, denoted by this special URI:

```
DELETE api/sso/user/multi HTTP/1.1
```

And the body content is:

NOTE: Some debate whether body content should be sent with a DELETE request. It may not be possible in all clients, and it is does not really conform with REST. This form of the request is supported for those that can do so; other clients can send separate DELETE requests to log out multiple users.

JSON

```
[
     { "ip": "ip-address-1", "name": "user-name", "domain":
     "domain-name", "reason": "text" },
     { "ip": "ip-address-2", "name": "user-name", "domain":
     "domain-name", "reason": "text" }
]
```

XML

```
<users>

     <user>

          <ip>ip-address-1</ip>
          <name>user-name</name>
          <domain>domain-name</name>
          <reason>text</reason>

     </user>

     <user>

          <ip>ip-address-2</ip>
          …

     </user>

</users>
```

HTTP OPTIONS Support

The HTTP OPTIONS request is supported. This allows clients to query the methods supported and the parameters that can be sent with them, making the API self-describing. This also provides a mechanism for extensibility (refer to Backward Compatibility for more information).

The reply to an options request returns **200 OK** and includes this HTTP header:

```
Allow: POST,DELETE,OPTIONS
```

The body content is returned in JSON or XML format, per the request's **Accept** header (refer to Request Format). The content is formatted as shown in Response Format Options

User Attributes

The following user attributes are accepted in the user entry with a user login notification. Note that this API is extensible and additional attributes may be added to in future.

Term	Definition
ip, **ipv4**, **ipv6**	The IP address that the user will be sending traffic from (refer to IPv6 Support for more information).
name	The user's account name.

Term	Definition
type	One of the following: • **domain**. • **local-trusted**. A non-domain user who had gone through a sufficient level of authentication that they can be trusted. The user is made a member of the Trusted Users user group on the SonicWall firewall. • **local-untrusted**. An untrusted non-domain user who will not be made a member of the Trusted Users user group. If type is not given, the default is determined as follows: • If a **domain** attribute is given, the firewall assumes the user is a domain user. • If a **domain** attribute is not given, but LDAP is enabled, LDAP is used to look the user up. • If a **domain** attribute is not given and LDAP is not enabled, the firewall assumes the user is an untrusted, non-domain user.
domain	The domain name for a domain user. Can be either the full DNS name or, for Windows users, the short-format NetBIOS name. If this is not given for a domain user then it will be looked up via LDAP if that is enabled, otherwise an error is returned.
group	The name of a user group to which the user has membership, if that is known. There can be multiple of these and they can specify all of the user's memberships or additional memberships to set in addition to the user's memberships in the domain.
group-lookup	Whether user groups (additional user groups if any are given in the message) should be looked up via LDAP. Values are **true** or **false**, and if not present then the default is determined as follows: For domain users: • If no groups are given in the message, then look them up. • If any groups are given in the message, then do not look up additional ones. For non-domain users, do not look up user groups. (This is silently ignored if LDAP is not enabled.)
ip-remote	This is for an unusual situation where the authentication server is on the "outside" of the firewall and the user IP addresses that it sends are external/remote ones that may need to be translated to the "inside" user IP addresses that the SonicWall will be receiving traffic from. Values are **true** or **false**, and if not present then the default is false. A typical example requiring this would be for a firewall at a remote site which forwards user authentication to a central site via VPN. If NAT were used on the VPN tunnel then the user IP addresses returned from the central site's authentication server would be the remote IP addresses, and the SonicWall would need to do a NAT "reverse-mapping" to translate those back to the users' actual IP addresses at the remote site. Note this would require that 1:1 NAT be used. **NOTE:** Refer to User Logout for an alternative way of specifying this in a DELETE request

IPv6 Support

This API supports receiving login/logout notifications for IPv6 users. To do this without needing to decode and classify the IP address in every request, SonicWall products that support users with IPv6 addresses accept three alternative fields for IP addresses as described in the following:

	Replace this:	With one of these:

JSON	`"ip": "`*`ip-address`*`"`	`"ip": "ip-address"`
		`"ipv4": "ipv4-address"`
		`"ipv6": "ipv6-address"`

XML	`<ip>`*`ip-address`*`</ip>`	`<ip>`*`ip-address`*`</ip>`
		`<ipv4>`*`ipv4-address`*`</ipv4>`
		`<ipv6>`*`ipv6-address`*`</ipv6>`

In the first form, where the IP version is not specified, the SonicWall firewall derives the version from the content. By specifying the appropriate version, you can avoid the small overhead of having the firewall derive it.

Responses

On success the SonicWall will return an HTTP 200 OK response with an empty body. On failure the following status codes from RFCs 7231and 4918 are returned:

Responses	Definition
General:	
207 Multi-Status	For failure of a multi-user POST or DELETE (see below).
400 Bad Request	The request content is not valid.
401 Unauthorized	The request authenticator failed validation.
403 Forbidden	Not using HTTPS (which is always required).
405 Method Not Allowed	Something other than a POST, DELETE or OPTIONS was sent.
406 Not Acceptable:	Data is to be returned but the request doesn't accept either XML or JSON.
415 Unsupported Media Type	The content type is something other than XML or JSON.
500 Internal Server Error	Failed to add or delete the user for some reason other than below.
Specific to a POST request:	
409 Conflict:	Unable to replace an existing user. For example if the user name is already in use when "Enforce login uniqueness" is enabled.
414 Request-URI Too Long	Something follows "user/sso" in the URI of a POST.
422 Unprocessable Entity	Unable to log the user in for some other reason. For example if an invalid user name (e.g. "admin") is given.
503 Service Unavailable	Too many users are concurrently logging in. **NOTE:** This message is displayed when the server is unable to handle the request due to a temporary overload. The overload will likely be alleviated after some delay.
507 Insufficient Storage:	User limit exceeded

Responses	Definition
Specific to a DELETE request:	
404 Not Found:	No user is logged in from the given IP address.

When certain status codes are returned, HTTP headers and/or body content are included in the response in accordance with RFC 7231. The response contains information about the error within the body content, if present, and is formatted as follows:

JSON

```
{ "error ": "description of the error" }
```

XML

```
<info>
     <error>description of the error</error>
</info>
```

The response codes for which this is present includes **400 Bad Request**, **409 Conflict**, **422 Unprocessable Entity**, **503 Service Unavailable** and **507 Insufficient Storage**.

In certain rare cases, body content giving some informative message may be included with the **200 OK** returned on success. For example, in the reply to a POST, if the given user is already logged in, it returns **200 OK** with a message in the body reporting that. The format for this is:

JSON

```
{ "message": "some informative message" }
```

XML

```
<info>
     <message>some informative message</message>
</info>
```

For a multi-user POST or DELETE, a **200 OK** status means that all users were added or deleted. If any user fails to get added or deleted then **207 Multi-Status** is returned, and the XML or JSON in the body indicates the result for each individual user as follows (confirming with RFC 4918 section 13). Each ***status-string*** is an HTTP status line string with a status code and reason phrase from the table above (for example, **HTTP/1.1 404 Not Found**).

JSON

```
"multistatus": [
     { "href": "api/sso/user/ip-address-1", " status": "status-string",
     "error": "description" },
     { "href": "api/sso/user/ip-address-2", " status": "status-string",
     "error": "description" }
]
```

XML

```
<multistatus>
     <response>
```

```
            <href>api/sso/user/ip-address-1</href>
            <status>status-string</status>
            <error>description of the error</error>

        </response>

        <response>

            <href>api/sso/user/ip-address-2</href>
            <status>status-string</status>
            <error>description of the error</error>

        </response>

    </multistatus>
```

The error description fields may or may not be present depending on the status code. They may be replaced by "message" fields as above with a **200 OK** status.

Persistent Connections

The SSO API is configurable to allow persistent connections from an HTTP 1.1 client. When this is enabled for a client (it is disabled by default), the client is able to keep connections open across multiple requests. Closing a connection then becomes the responsibility of the client, which it can do. Or it can request the SonicWall firewall to close the connection after completing a request by including a **Connection: close** header in the request. If the **Connection: close** header is present, the SonicWall firewall closes an HTTP 1.1 connection after returning its reply; otherwise, it does not.

This only works if the client is using HTTP 1.1. Using the HTTP 1.0 **Connection: Keep-Alive** header for this is not supported.

A connection is not kept open if the request is not authenticated. This is to avoid keeping connections open to a possible malicious attacker.

If allowing persistent connections is not enabled, the SSO API operates in the traditional HTTP 1.0 way with a separate connection for every HTTP request. The SonicWall firewall closes each connection after returning its reply. Since HTTP 1.1 connections are persistent by default, if this is not enabled then all replies to HTTP 1.1 requests include the **Connection: close** header to notify the clients of this, in accordance with RFC 2616 section 14.10.

NOTE: If large numbers of clients send requests to a SonicWall firewall, you shouldn't use persistent connections to avoid over-use of its limited file descriptors. If an API client sends requests to a large number of SonicWall firewalls, it does not want to keep connections open to all of them. But in a case where one client sends to a single SonicWall firewall or to a small number of them, and where large numbers of requests are sent during peak times, using a persistent connection could be advantageous.

3

Security and Authentication

The SSO API runs over HTTPS and is inherently secure from external snooping. For additional security, the two different directions have different requirements:

1. The SonicWall firewall must be able to verify that API requests are being sent from the correct source; otherwise, a hacker could potentially give themselves access through the firewall by impersonating the API client and spoofing requests from it to get logged in.
2. The client should also be able to verify that its API requests are being sent to the SonicWall firewall and not to something impersonating it. This is less import since the requests do not contain any highly sensitive information—mainly just the users' names and locations (no passwords), for example.

NOTE: Operation over HTTP is not allowed.

These two security requirements are addressed as follows, and as detailed in the following sections:

1. To validate the client's identity, the host name or Ip address needs to be configured on the SonicWall firewall. In addition, a valid client-side certificate or using a shared secret (or both) should be required.
2. To validate the SonicWall's identity by the client, if needed, you need to use a valid HTTPS certificate on the SonicWall firewall. It can also be partially addressed by use of a shared secret (more details are provided in the following sections).

IMPORTANT: If the priority is to guard against illicit access to potentially sensitive data through the SonicWall firewall, the security recommendations in this chapter should be followed carefully.

Client IP Address Configuration

Each client allowed to use the SSO API must be configured on the SonicWall firewall, giving their IP address or DNS host name. Any requests received from other IP addresses are silently ignored.

SSL Certificates

The SonicWall firewall is configurable to require a valid certificate from the sender in the API requests. For this, the certificate supplied by the client must be verifiable on the SonicWall firewall (that is, the relevant CA certificate must be imported if not built-in) and either its subject name must match the host name configured for the client, or the host name must be given in its subject alternate names.

NOTE: If a client certificate is used, it must be a host certificate that is issued for the client's host name, as configured. That usually requires the client be configured by DNS name. A user certificate (one issued to a user name) cannot be used in this situation.

The SonicWall firewall uses its HTTPS management certificate for its replies on the API, which, by default, is a self-signed certificate. Installing a valid (non-self-signed) certificate on the SonicWall firewall and having the client validate it is optional.

Request Authentication with Shared Secret

Using Shared Secret Authentication is optional but highly recommended. If used, each client has a shared secret key configured, and an authenticator derived from that is passed in requests, and optionally, in replies as well. Three levels of authentication are configurable per-client on the SonicWall firewall:

Level of Authentication	Description
High	The client's authenticator value is calculated using a mechanism that makes it different in each request and response, hence guarding against the possibility of a replay attack. With this level of authentication, you can choose to use SHA256, SHA512, or allow both, plus you can support negotiation of this with the client. **NOTE:** Negotiating the level of access prevents a different user from getting access by replaying the same authenticator in a new request. On its own, this does not prevent getting repeat access for the same user, from the same IP address, by replaying the exact same request. That is generally prevented by operating over HTTPS since that prevents external snooping and replay is not possible without being able to snoop the messages. To be fully safe from this, requires using the CSRF prevention mechanism. Refer to CSRF Prevention for more information.
Medium	The client's authenticator value is calculated using a SHA256-based mechanism. The SHA256-based mechanism allows the same value to be sent in all requests. This level of authentication is aimed at ease of testing, for example, when developing the client-side scripts. It allows a pre-calculated authenticator to be used, but it is not recommended for production use.
Low	No shared secret based authentication of requests or replies is performed, relying only on validating the sender's IP address. This is mainly supported as an ease-of-use option during development of the scripts or for debugging

ⓘ **IMPORTANT:** The medium and low security levels are intended for testing or diagnostic purposes only. They are not recommended for production use. High level of authentication should always be used where security may be an issue.

This authentication mechanism utilizes the HTTP **Authorization** header field, formatted as follows:

```
Authorization: SNWL-API-Auth base64-coded-authenticator
```

ⓘ **NOTE:** The authentication mechanism conforms with RFC 2617.

Security Level Negotiation

Some limited security level negotiation is supported when using high level security. It allows for multiple API clients and/or multiple SonicWall firewalls that may be configured differently in this respect (possibly during a multi-step update procedure). If a SonicWall firewall receives a request with an authenticator of the wrong length for the mechanism(s) that it is configured for then it returns a **401 Unauthorized** status and includes a **WWW-Authenticate** header in the reply as follows:

```
WWW-Authenticate: SNWL-API-Auth Hash:(SHA256|SHA512|SHA256,SHA512)
```

If the SonicWall firewall is configured to allow only SHA256 and it receives a request using SHA512, it returns this message specifying **Hash:SHA256**, and vice-versa. If it is configured to allow both SHA256 and SHA512 and it receives something different, it returns the message specifying **Hash:SHA256,SHA512**. It is then up to the client to decide whether it complies and re-sends the request with a new, recalculated authenticator.

NOTE: Using medium level security has no negotiation. If a client tries to use medium level to a SonicWall firewall that is configured for high level with SHA512, it gets the response shown above. The SonicWall firewall uses the authenticator length to differentiate SHA256 from SHA512 and so cannot distinguish medium level from high with SHA256.

Authenticator Format

For the different security levels, the authenticator is calculated as described in the following sections. Fields are transmitted from left to right (octet number 0 first) and bit numbering is from most to least significant; that is, bit 0 is the most significant bit.

High Level Security with SHA256

Setting the authentication level to high with SHA256 is generally recommended in a production environment.

Request from the client:

The authenticator in the Authorization header consists of base-64 encoding of a 64-octet sequence that is formatted:

0 3	4 7	8 31	32 63
Flags	Seq-num	Rqst-nonce	Hash

Where:

Flags	A 4-octet field of bit flags:	
		Bits 0-30: Reserved; they must be set to zero.
		Bit 31: Set to 1 if the client expect an authenticator in the reply; set to 0 otherwise.
Seq-num	A 4-octet sequence number used only when CSRF Prevention is enable; otherwise it is ignored.	
Rqst-nonce	A 24-octet random number that should be separately generated by the client, for each request.	
Hash	A 32-octet number that is computed as a SHA256 hash of the following values:	
		The flags, seq-num and rqst-nonce fields.
		The shared secret key.
		The full request body content or the URI for a request with no content.

Reply from the SonicWall:

When requested via the flags in the request authenticator, the authenticator in the **Authorization** header of the reply consists of a base-64 encoding of a 64-octet sequence that is formatted:

0 31	32 63
Resp-nonce	Hash

Where:

Resp-nonce	A 32-octet random number randomly generated by the firewall for each reply.
Hash	A 32-octet number that is computed as a SHA256 hash of the following values:
	The 64-octet authenticator from the request (not base-64 encoded).
	The 32-octet resp-nonce value.
	The shared secret key.

You decide whether or not to validate the reply authenticator. Validating the reply authenticator on its own does not guard against the possibility of sending requests to a malicious device impersonating the SonicWall firewall since it is checked after-the-fact. Validation of it does at least generate warnings of potential issues. If protecting the user names and locations from possible snooping is considered highly important then a certificate must be used for that.

If you are not going to validate the reply authenticator, set the flag for requesting it to 0 to avoid having the SonicWall firewall spend unnecessary overhead calculating it.

High Level Security with SHA512

When the authentication level is set to high level with SHA512 achieves an extremely high level of security but at some performance penalty versus using SHA256:

Request from the client:

The authenticator in the Authorization header consists of a base-64 encoding of a 128-octet sequence that is formatted:

0 3	4 7	8 63	64 127
Flags	Seq-num	Rqst-nonce	Hash

Where:

Flags	A 4-octet field of bit flags:
	Bits 0-30: Reserved; they must be set to zero.
	Bit 31: Set to 1 if the client expect an authenticator in the reply; set to 0 otherwise.
Seq-num	A 4-octet sequence number used only when CSRF Prevention is enable; otherwise it is ignored.
Rqst-nonce	A 56-octet random number that should be separately generated by the client, for each request.
Hash	A 64-octet number that is computed as a SHA512 hash of the following values:
	The flags, seq-num and rqst-nonce fields.

		The shared secret key.
		The full request body content or the URI for a request with no content.

Reply from the SonicWall:

When requested via the flags in the request authenticator, the authenticator in the **Authorization** header of the reply consists of a base-64 encoding of a 128-octet sequence that is formatted:

0	63	64	127
Resp-nonce		Hash	

Where:

Resp-nonce	A 64-octet random number randomly generated by the firewall for each reply.	
Hash	A 64-octet number that is computed as a SHA512 hash of the following values:	
		The 128-octet authenticator from the request (not base-64 encoded).
		The 64-octet resp-nonce value.
		The shared secret key.

You decide whether or not to validate the reply authenticator. Validating the reply authenticator on its own does not guard against the possibility of sending requests to a malicious device impersonating the SonicWall firewall since it is checked after-the-fact. Validation of it does at least generate warnings of potential issues. If protecting the user names and locations from possible snooping is considered highly important then a certificate must be used for that.

If you are not going to validate the reply authenticator, set the flag for requesting it to 0 to avoid having the SonicWall firewall spend unnecessary overhead calculating it.

Medium Level Security

When the authentication level is set to medium for ease of testing:

Request from the client:

The authenticator in the **Authorization** header is the same as with the high level, except for:

Where:

Rqst-nonce	A 24-octet random number randomly generated that may be repeated across multiple requests to the same authenticator in each.	
Hash	A 32-octet number that is computed as a SHA256 hash of the following values:	
		The flags, seq-num and rqst-nonce fields.
		The shared secret key.

Reply from the SonicWall:

When requested via the flags in the request authenticator, the authenticator in the **Authorization** header is computed in the same way as with high-level security, except for:

Resp-nonce	A 32-octet random number that is computed as a SHA256 hash of the rqst-nonce and the firewall's 6-byte MAC address.

Note that this medium level shared-secret authentication is intended to allow manual testing by sending requests from a utility, such as Linux curl, without needing to re-calculate a separate authenticator for each, while still exercising nearly all the authentication mechanisms used with the high level. The tester can generate a nonce value and calculate the authenticator from that, then use those same values throughout a test session. This could be useful when developing client-side scripts for the API.

Note also that CSRF prevention operates with medium level security, but using them together would be rather self-defeating. The CSRF function is to prevent replay but the main purpose of medium level as implemented here is to allow reuse of authenticators for testing.

CSRF Prevention

The authentication mechanism described previously avoids request replay by external snooping since HTTPS helps prevents snooping. However, on its own, it is not 100% secure from malicious software that could get itself installed on the API client's host computer. Such malware could theoretically snoop the client's requests before the SSL encryption and then replay them directly from the client machine. This is known as Cross Site Request Forgery (CSRF) and the technique for preventing it in a non-session-based situation such as this SSO API is known as Synchronizer Token Pattern (STP).

To achieve STP for CSRF prevention, in addition to the mechanism already detailed above, you need to also include a value that is unique to every request, including for a re-send of the same request should that be necessary. For this, use a sequence number in the request.

Using a sequence number is optional since it does add some complexity to operation of the API, adding in an occasional handshake between the client and the SonicWall firewall. If it is enabled on the firewall, then it must be used by the client (but not vice-versa).

IMPORTANT: If the client that is sending requests over the SSO API consists of software or scripts that are running on a computer that could potentially be at any risk of getting malware installed, then it is recommended that CSRF prevention be enabled.

When the CSRF prevention mechanism is enabled it operates as follows:

1 The seq-num field in the client's request authenticator must be set to a 32-bit number that is incremented sequentially in every request sent to a particular SonicWall firewall.

2 When the SonicWall firewall receives a request, it validates the authenticator and checks the sequence number. It expect the sequence number to be one greater than that of the previous request received from the client. If the sequence number is as expected then the request is accepted.

3 Should the sequence numbering get out of step (for example, if the client or the SonicWall firewall is restarted) then the sequencing can be re-synchronized by having the SonicWall firewall invoke a handshake mechanism if it receives a request with a sequence number that is not as expected. The firewall then returns a new randomly-generated sequence number value to the client who must use then use that in its next request. (The client should really re-send the same request with this new sequence number). A **401 Unauthorized** status is returned and a **WWW-Authenticate** header is included in the reply, formatted as follows:

   ```
   WWW-Authenticate:  SNWL-API-Auth  Reset:new-sequence-number
   ```

   The new sequence number value is passed back as a decimal number in plain text.

When setting up CSRF Prevention, be aware of the following:

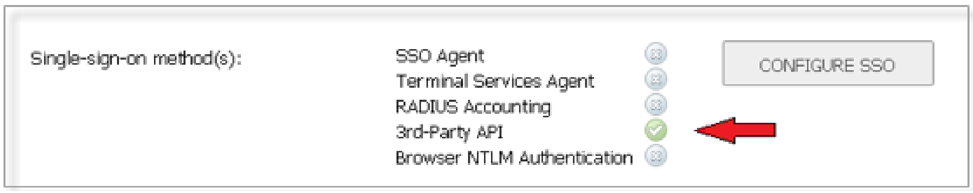
- The sequence number reset mechanism also acts to synchronize it at initial startup. The client can send any number that it wishes to start from in its first request, and if that is not what the SonicWall firewall is expecting, this mechanism is invoked, passing a new sequence number to the client which can then re-send the request with that new number.

- After a reboot, the SonicWall firewall starts with the expected sequence number for each client initialized to 1. If the clients likewise start by sending sequence number 1 when they start up then no synchronization handshake is needed if both are restarted together.
- The sequence number reset mechanism is clear text (before the HTTPS encryption), and the next sequence number is easily predictable. This does not result in an insecurity because the sequence number is included in the calculation of the authenticator hash; hence, it is not possible to construct a valid request with it unless the shared secret is known.
- When CSRF Prevention is enabled, each API client must send its requests to a firewall sequentially, waiting for the reply to each before sending the next. Otherwise, if overlapping requests were sent then they could be received out of order at the SonicWall firewall. If necessary, a multi-threaded client should use some form of locking to avoid two threads sending simultaneous requests to the same SonicWall firewall.
- If an authenticator fails validation then the next expected sequence number is not incremented, so the client should not increment its sequence number for the next request after a **401 Unauthorized** response. Doing this avoids an attempted CSRF attack from interfering with the sequence numbering of the real requests from the client.

A

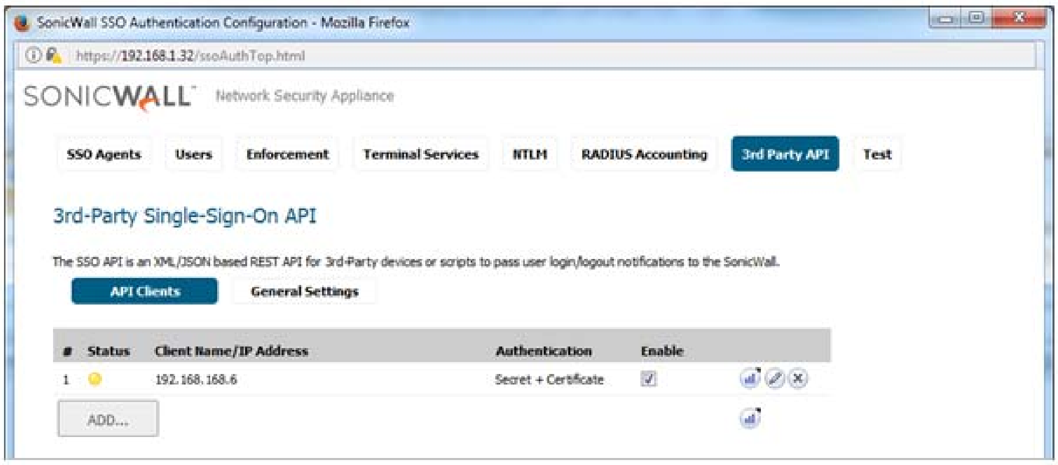
Configuration

A checkmark icon enables this SSO API along with the other SSO mechanisms on the **MANGAGE** view under **System Setup | Users / Settings**. It shows checked / green when it is enabled:


Single-sign-on method(s):

SSO Agent	
Terminal Services Agent	
RADIUS Accounting	
3rd-Party API	✓
Browser NTLM Authentication	

The API can be enabled either by clicking that icon, or by a checkbox in the SSO configuration dialog. To configure the API and its clients click the **CONFIGURE SSO** button to open the SSO configuration dialog and select the 3rd Party API tab:

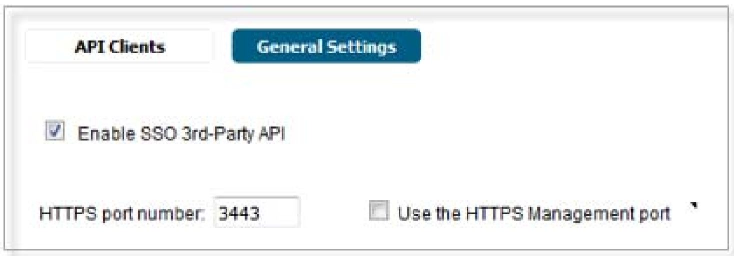

SonicWall SSO Authentication Configuration - Mozilla Firefox
https://192.168.1.32/ssoAuthTop.html
SONICWALL Network Security Appliance
SSO Agents
Users
Enforcement
Terminal Services
NTLM
RADIUS Accounting
3rd Party API
Test
API Clients
General Settings
ADD...

3rd-Party Single-Sign-On API

The SSO API is an XML/JSON based REST API for 3rd-Party devices or scripts to pass user login/logout notifications to the SonicWall.

#	Status	Client Name/IP Address	Authentication	Enable
1	●	192.168.168.6	Secret + Certificate	✓

The **General Settings** tab includes a checkbox to enable/disable the API, plus configuration of its HTTPS port number with an option to use the same port as for HTTPS management:

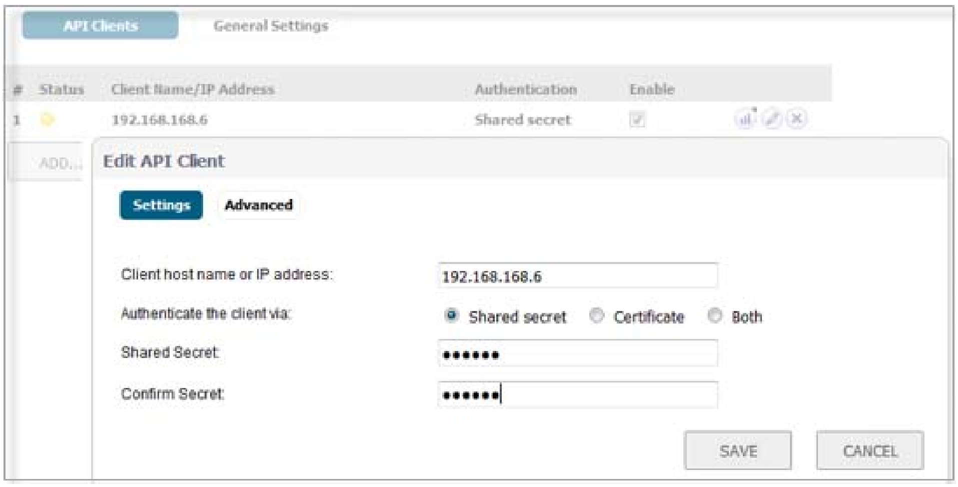


[✓] Enable SSO 3rd-Party API

HTTPS port number: 3443 [ ] Use the HTTPS Management port

NOTE: In larger installations where the SSO API could get very heavily used, you should dedicate a port number since that could help avoid floods of requests on the API from interfering with web management. If client certificates are used with the API, a dedicated port number *must* be used.

The list of API clients here follows the same model as the other SSO agents and clients in this dialog, allowing quick edits to some fields by clicking on them in the list. Click **ADD…** to add a new client, or click the Edit icon of an existing client to open this window for changes:



#	Status	Client Name/IP Address	Authentication	Enable
1	●	192.168.168.6	Shared secret	✓

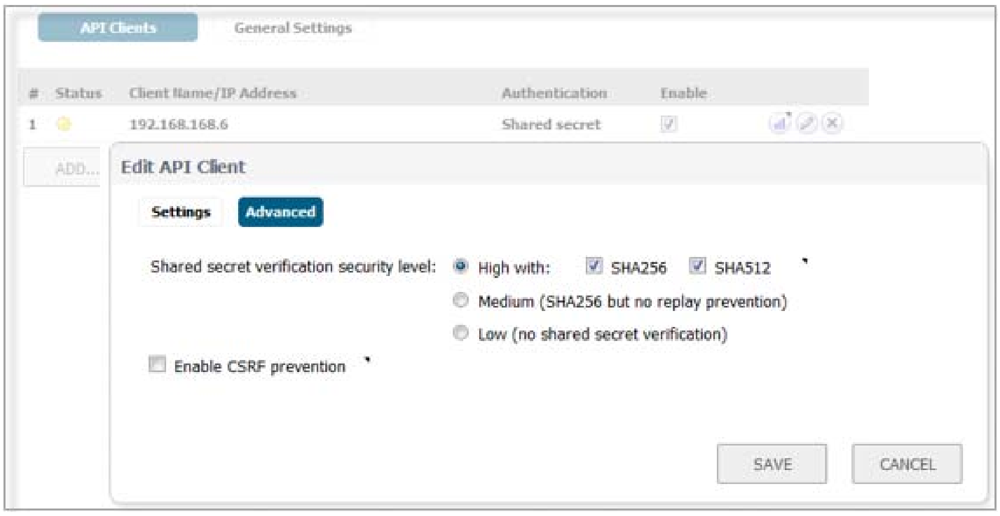
Edit API Client

Settings | Advanced

Client host name or IP address:	192.168.168.6
Authenticate the client via:	[✓ Shared secret] Certificate Both
Shared Secret:	••••••
Confirm Secret:	••••••

Note that the options to select Certificate or Both for authentication of the client is only available if a dedicated HTTPS port number is used.

The **Advanced** tab has the options for overriding the default security level and for enabling CSRF prevention:



#	Status	Client Name/IP Address	Authentication	Enable
1	●	192.168.168.6	Shared secret	✓

Edit API Client

Settings | Advanced

Shared secret verification security level: [✓ High with: [✓ SHA256] [✓ SHA512]]
Medium (SHA256 but no replay prevention)
Low (no shared secret verification)

☐ Enable CSRF prevention

IMPORTANT: The medium and low security levels are intended for testing or diagnostic purposes only and are not recommended for production use. High levels should always be used where security may be an issue. If the client sending requests over the SSO API consists of software or scripts that are running on a computer potentially at risk of getting malware installed, CSRF prevention should be enabled.

B

Response Format Options

The following table shows the content returned in the reply to an OPTIONS request. It describes the supported HTTP methods and their content. The overall structure—described here in Augmented Backus–Naur Form, ABNF—is:

options-content	= *(method)	*; List of the supported HTTP methods*
method	= name description parameters	
parameters	= *(parameter)	*; List of the parameters that can be sent with a method, or the parameters that can be included in a container*
parameter	= (container-param / attribute-param)	
container-param	= name "container" description parameters	*; Describes a user or list of users*
attribute-param	= name param-type description param-attributes	*; Describes a user attribute*
param-type	= ("string" / "integer" / "boolean" / "ip address" / "enum")	
param-attributes	= *(param-attribute)	
param-attribute	= (required / multi-instance / values)	
required	= (true / false)	*; true if the parameter must be included*
multi-instance	= (true / false)	*; true if the parameter can be included more than once*
values	= *(value)	*; List of the allowed values for param-type enum*

NOTE: This is rather loose ABNF and does not attempt to nail down the exact format of each component, which can be seen from the XML and JSON content that follows. This ABNF definition is just trying to clarify the structure of the data.

The body content that is returned is as follows:

JSON

```
{
  "methods": {
    "POST": {
      "description": "To notify of login of a user or multiple users",
      "user": {
        "description": "Specifies a single user",
        "ip": { "type": "ip address", "required": true },
        "name": { "type": "string", "required": true },
        "domain": { "type": "string", "required": false },
        "type": { "type": "enum",

          "values": [ "domain", "local-untrusted", "local-trusted",
          "guest" ], "required": false },

        "groups": [

          { "name": "group", "type": "string", "required": false }

        ],
        "group-lookup": { "type": "boolean", "required": false }

      },
      "users": [

        {

          "object": "user",
          "description": "For multiple users, specifies one user as
          above"

        }

      ]

    },
    "DELETE": {

      "description": "To notify of logout of a user or multiple users",
      "user": {

        "description": "Optional data for logging when the URI selects a
        single user",
        "criteria": "The URI must give the user's IP address",
        "name": { "type": "string", "required": false },
        "domain": { "type": "string", "required": false },
        "reason": { "type": "string", "required": false }

      },
      "users": [

        {

          "object": "user",
          "description": "For multiple users, specifies one user",
          "criteria": "The URI must specify 'api/sso/user/multi'",
          "ip": { "type": "ip address", "required": true },
          "name": { "type": "string", "required": false },
          "domain": { "type": "string", "required": false },
          "reason": { "type": "string", "required": false }

        }

      ]

    }

  }

}
```

XML

```
<options>
  <methods>
    <method>
      <name>POST</name>
      <description>To notify of login of a user or multiple
      users</description>
      <parameters>
        <parameter>
          <name>user</name>
          <type>container</type>
          <description>Specifies a single user</description>
          <parameter>
            <name>ip</name>
            <type>ip address</type>
            <required>true</required>
          </parameter>
          <parameter>
            <name>name</name>
            <type>string</type>
            <required>true</required>
          </parameter>
          <parameter>
            <name>domain</name>
            <type>string</type>
            <required>false</required>
          </parameter>
          <parameter>
            <name>type</name>
            <type>enum</type>
            <values>domain, local-untrusted, local-trusted,
            guest</values>
            <required>false</required>
          </parameter>
          <parameter>
            <name>group</name>
            <type>string</type>
            <required>false</required>
            <multi-instance>true</multi-instance>
            <info>These can optionally be enclosed in a <groups>
            container</info>
          </parameter>
          <parameter>
            <name>group-lookup</name>
            <type>boolean</type>
            <required>false</required>
          </parameter>
        </parameter>
        <parameter>
      <name>users</name>
      <type>container</type>
      <description>For multiple users, holds a set of user
      entries</description>
      <parameter>

        <name>user</name>
        <type>container</type>
        <description>For multiple users, specifies one user as
        above</description>

      </parameter>

    </parameter>

  </parameters>

</method>
<method>

  <name>DELETE</name>
  <description>To notify of logout of a user or multiple
  users</description>
  <parameters>

    <parameter>

      <name>user</name>
      <type>container</type>
      <description>Optional data for logging when the URI selects a
      single user</description>
      <criteria>The URI must give the user's IP address</criteria>
      <parameter>

        <name>name</name>
        <type>string</type>
        <required>false</required>

      </parameter>
      <parameter>

        <name>domain</name>
        <type>string</type>
        <required>false</required>

      </parameter>
      <parameter>

        <name>reason</name>
        <type>string</type>
        <required>false</required>

      </parameter>

    </parameter>
    <parameter>

      <name>users</name>
      <type>container</type>
      <description>For multiple users, lists users who have logged
      out</description>
      <criteria>The URI must specify
      'api/sso/user/multi'</criteria>
      <parameter>

        <name>user</name>
        <type>container</type>
        <description>For multiple users, specifies one
```

```
          user</description>
          <parameter>

            <name>ip</name>
            <type>ip address</type>
            <required>true</required>

          </parameter>
          <parameter>

            <name>name</name>
            <type>string</type>
            <required>false</required>

          </parameter>
          <parameter>

            <name>domain</name>
            <type>string</type>
            <required>false</required>

          </parameter>
          <parameter>

            <name>reason</name>
            <type>string</type>
            <required>false</required>

          </parameter>

        </parameter>

      </parameter>

    </parameters>

  </method>

</methods>
```

C

SonicWall Support

Technical support is available to customers who have purchased SonicWall products with a valid maintenance contract and to customers who have trial versions.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. To access the Support Portal, go to https://www.sonicwall.com/support.

The Support Portal enables you to:

- View knowledge base articles and technical documentation
- View video tutorials
- Access MySonicWall
- Learn about SonicWall professional services
- Review SonicWall Support services and warranty information
- Register for training and certification
- Request technical support or customer service

To contact SonicWall Support, visit https://www.sonicwall.com/support/contact-support.

About This Document

Legend

⚠ **WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.**

⚠ **CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

ⓘ **IMPORTANT, NOTE, TIP, MOBILE, or VIDEO:** An information icon indicates supporting information.

SSO API Reference Guide
Updated - October 2017
232-004084-00 Rev A