# Automated Configuration Generation for a Full-Mesh VPN

Cisco ASA device configuration involves the endless repetition of a small number of key pieces of information; a group of devices which participate in a fully-meshed VPN will have configurations which are very similar, with a few important details which must be properly reciprocated between peers. *Sitemesh* automates this process by expanding a template to generate a matching set of device configuration files.

## Syntax

A ***sitemesh*** template is a flat text file with two basic declaration types: *lines*, which are a simple line of text, and *blocks*, which are multiple lines enclosed by braces. Any given keyword must be unique within a device section; with some exceptions, keywords must be unique within the enclosing block. Note that the parser is simple-minded, so a block must be identified by exactly two words, with the open-brace on the same line.

Comments and blank lines will be ignored, where "comment" is any line which begins with any of octothorpe (#), semicolon (;) or double-slash (//). Comments that do not begin a line will have strange effects.

Words which are not expanded by sitemesh will (usually) be passed through to the generated configuration. Because keywords must be unique, a semicolon will be expanded to multiple instances of the keyword; for example, "terminal width 120; pager 40" becomes "terminal width 120" and "terminal pager 40". Some blocks require multiple key-value pairs; these use a notation such as "hash=sha encryption=3des".

The magic word **exit** causes all processing to stop; this feature can be used to keep arbitrary notes, definitions for spare sites, etc, with the template. To insert a block of arbitrary native configuration, use the word **verbatim**, which takes as its one argument the word which indicates the end of the block.

Fatal errors will cause sitemesh to exit with a list of problems. Otherwise, one or more configurations are generated, along with a list of notices and/or warnings. While there is a great deal of sanity-checking logic, sitemesh is very "trusting", so it is possible to generate a bad configuration; this will usually include a set of "strange" warnings.

## Site

A *site* begins with the **site** keyword; parts of the template before the first site are used as globals for all sites. Every site must contain at least one *device*; this keyword triggers the creation of a configuration file. The *supernet* (shortest prefixes which define this site) and *tunnel* (crypto-map name) keywords are required for any site which will support VPN access.

The optional *deviceflags* keyword is used to enable special features. Setting this option to "ios" will cause sitemesh to generate a stub of IOS configuration for this site, rather than the default ASA syntax. By default, "pre-8.2" style ASA configuration syntax is used; to enable new "8.3 and above" syntax, set deviceflags to "objectnat".

## Globals

Default values can be declared before the first *site*; these will generally be merged with the equivalent values in the site block. The *default* blocks can **only** be declared as global; the *windows-dc* and *windows-map* blocks are declared with the name **default** in the global scope.

Because sitemesh is a multi-pass compiler, forward references are permissible, for instance when declaring a multi-site supernet with sites that haven't (yet) been defined.

Some options do not make sense as globals; a few will generate errors, the rest are silently ignored.

## Networks

A basic *network* consists of one or more CIDR address prefixes, each listed with the *address* keyword, all of which will be listed in an object-group declaration named *site_networkname*_**net**. If this network has a physical interface, the keywords *interface*, *security-level*, and *description* are passed through and a *nameif* directive is generated; if the interface unit number contains a period, the *vlan* directive is added, and the parent interface is automatically configured as a trunk.

A network may belong to one or more *netgroups* by means of the *group* keyword; these groups are named **nets_***groupname,* and are used as the destination addresses in an access-list.
Defaults for each network may be declared with a *default* block in the global scope. Site-specific keywords will generally override these defaults, with two exceptions: a more-specific *nat interface* statement will replace a default *nat 1 interface*, and any default *access* is appended to the site-specific *access*.

A default route can be specified with the *default* keyword; this has the side effect of flagging this interface as "outside" (regardless of its actual name) for VPN tunnels and exposed servers.

The *access* keyword may be specified more than once and will generate an *access-list* named *in_networkname* which is applied as an inbound access-group for that interface. One of the keywords *permit, deny,* or *block* (inverted *deny*) must follow *access*, after which a *netgroup* is specified, with an optional *port*. If the network block contains the *strict* keyword, this access-list will specify this network for the source, otherwise a source of *any* is used.

Outbound access-lists are generated with the *block* keyword, which always adds an explicit *permit any* at the end; these are generally used to prevent "leakage" of private address space at the egress interface.

The *nat* and *global* keywords are passed to the generated configuration and behave as expected; for the nat keyword, valid arguments are *any, interface,* and *nonat,* where *interface* is expanded to "the address and netmask of this interface".

Dialup VPN configuration is generated by including the *presharedkey* keyword. To restrict the VPN access with a split-tunnel, include the *split* keyword and specify a supernet; the split-tunnel-policy defaults to **tunnelspecified** unless changed with the *splittunnel* keyword. Many VPN-specific keywords are passed through to the device configuration.

When declaring additional *supernet* blocks, IP addresses or names can be used; names must either be other supernets, or networks in the local scope.

## Servers

A *server* is a named machine which generates an object-group declaration *site_machinename*_**host**. Like networks, servers may belong to one or more netgroups as enumerated in the *group* keyword.

Exposed servers are configured with the *public* keyword, which creates a static NAT to the *address.* The *outside* keyword lists the services to be provided; these are translated into an access-list (named *outside_servicename*) on the "outside" interface.

## Services

A *service* consists of one or more TCP or UDP port numbers which will be listed in an object-group declaration named *port_servicename*. Three keywords are understood in a service block: *type*, which must be "tcp", "udp" or "tcp-udp", *port*, which is any valid port number, and *include*, which inserts another service definition into this one.

## Options

Most *options* blocks contain device-native statements which are passed through to the generated configuration; these will be merged with a corresponding global block. Where an options block uses the *key=value* notation, these will be expanded into a multi-line stanza; for *options isakmp*, numbered keywords will generate *isakmp policy* statements, with defaults inherited from "policy 0".

Defaults listed in *options interface* will be applied to all physical interfaces.

The *ipsec* and *dynamic-map* blocks are similar to options blocks, except that the *blockname* is used as a map name.

For *ssh* and *telnet*, the *allow* keyword lists supernets for which ssh access should be allowed. Each device configuration will omit its own supernet unless the deviceoption "sshlocal" or "telnetlocal" is set. The *network* keyword lists one or more local networks for which access should be allowed; *address* specifies an IP network for which access should be allowed.

## LAN Tunnels

A *tunnel* block creates a tunnel between two sites. Because sitemesh generates the reciprocal configuration, only one end of the tunnel needs to be specified. The *key* keyword specifies a pre-shared key to be used for the tunnel. Sequence numbers in the generated crypto-map will be automatically created unless *deviceflags localcryptomap* is set, in which case the number in the *map* keyword will be used.

## Authentication

The *username* block is passed through to the device configuration.

VPN users can be validated against a Windows Active Directory domain by means of the *windows-dc* and *windows-map* blocks, each of which will inherit defaults from a matching global block named *default*. The windows-dc block describes a Domain Controller; the *host* keyword is matched to a network, and all others are copied into an aaa-server definition. The windows-map block defines a mapping of AD user groups to VPN policy groups; here, the *suffix* keyword is used for simple expansion of all other keywords in the block.

## IOS Devices

Enabling *deviceflags ios* causes a partial IOS-style configuration to be generated for the respective site. Access-lists for the crypto-map *networks* will be numbered from 140; these may each be overridden by specifying a *rule* keyword in the *tunnel* block. Also, *map* declarations are **always** used for local *tunnel* blocks, even when *localcryptomap* is not set.

Two access-lists are created for the crypto-map *access*; these can be configured with the *tunnelaccessin* and *tunnelaccessout* keywords, which accept an optional access-list number followed by an IP address and zero or more port numbers.

Note that IOS support is not robust; this feature should be used for small leaf nodes, with all tunnels defined on the IOS site.

## ASA Devices

Enabling *deviceflags objectnat* generates “modern” ASA 8.3+ configuration for the respective site: outside-facing access-lists use internal addresses, the new nat syntax is used, and VPN statements are ikev1. Future releases will support ikev2 and DAP.

## Example

Below are some configuration fragments along with a list of corresponding ASA statements. Generally, a sitemesh template implies any and all relevant ASA configuration.

```
site utah                           (sets the global site prefix to utah)
nonat 10.10.0.0/16 192.168.0.0/18   access-list nonat (for all combinations of these nets)

network dmz {                       object-group network utah_dmz_net
      interface Ethernet1           interface Ethernet1
      address 192.168.15.1/24       object-group network nets_staging
      security-level 10             access-list in_dmz
      group staging                 access-group in_dmz in interface dmz
      nat 0 nonat                   access-list out_dmz
      nat 50 interface              access-group out_dmz out interface dmz
      access deny unlikely          nat 0 access-list nonat
      access permit any             nat 50 192.168.15.0 255.255.2555.0
      block unlikely
}

network remote {                    object-group utah_remote_net
      address 192.168.0.0/22        ip local pool ip_remote
      presharedkey vpn-secret-key   access-group in_remote
      access permit corp            tunnel-group remote
      access deny unlikely          group-policy remote
      split utah
}

tunnel california {                 access-list location_california
      key secret-key                crypto map
}                                   tunnel-group

server webserver  {                 object-group network utah_webserver_host
      address 192.168.1.0/24        object-group network utah_webserver_public
      public 172.29.29.9            object-group network outside_web
      outside web
}
```