Advanced option settings on the command line

Set the interface and ports for the OpenVPN daemons

In the Admin UI under Server Network Settings there's the option to set the specific interface that the OpenVPN daemons should listen on. These are the programs that handle any incoming OpenVPN tunnel connections. This can for example be set to **all** so that it will simply listen to all available network interfaces, and usually this is the default. But in some situations it is desirable to configure the OpenVPN daemons to listen on a specific network interface only. You can also change the ports that the OpenVPN daemons listen on here, but we normally recommend not changing these unless there are unique circumstances.

Please note that the OpenVPN daemons and the web services are connected in a way. By default the OpenVPN Access Server comes configured with OpenVPN daemons that listen on port 1194 UDP, and OpenVPN daemons that listen on port 443 TCP. While the best connection for an OpenVPN tunnel is via the UDP port, we implement TCP 443 as a fallback method. It is likely that if you are on a public network that Internet connectivity is restricted. But TCP 443 is the port used for HTTPS traffic, and a lot of websites use HTTPS by default. So by having an OpenVPN TCP daemon on port TCP 443, chances are that even on such a restricted network your OpenVPN client program will be able to make a connection to the OpenVPN Access Server using the TCP fallback. It's no guarantee since some firewalls are quite sophisticated and can see the difference between a web browser and an OpenVPN client program, but it works on most simple firewalls.

As mentioned, TCP 443 is also used for HTTPS traffic, which the web interface of the OpenVPN Access Server also uses. You cannot have 2 different processes listening on the same port on the same server, so we use something we call service forwarding or port sharing. When you open the web interface of the Access Server on its default TCP port 443, the OpenVPN TCP daemon sees that request and recognizes that it is a browser request. It then internally redirects the traffic to the web services which are actually running on port TCP 943. When you change which interface the OpenVPN daemons listen on, you could be inadvertently denying yourself access via this port forwarding method. The solution then is to use the port that the web services are actually running on; TCP 943. To access the web interface at that port put :943 in the URL like so:
https://your.vpnserver.com:943/

Warning: changing these values may mean you have to reinstall your clients in order to be able to make a connection again, as these settings do not update automatically on the clients.

To set the interface name that the OpenVPN daemons should listen on:

```
./sacli --key "vpn.daemon.0.server.ip_address" --value <INTERFACE> ConfigPut
./sacli --key "vpn.daemon.0.listen.ip_address" --value <INTERFACE> ConfigPut
./sacli start
```

To set a specific port for the UDP OpenVPN daemons:

```
./sacli --key "vpn.server.daemon.udp.port" --value <PORT_NUMBER> ConfigPut
./sacli start
```

To set a specific port for the TCP OpenVPN daemons:

```
./sacli --key "vpn.server.daemon.tcp.port" --value <PORT_NUMBER> ConfigPut
./sacli start
```

To restore the default so it listens to all interfaces and ports TCP 443 and UDP 1194:

```
./sacli --key "vpn.daemon.0.server.ip_address" --value "all" ConfigPut
./sacli --key "vpn.daemon.0.listen.ip_address" --value "all" ConfigPut
./sacli --key "vpn.server.daemon.udp.port" --value "1194" ConfigPut
./sacli --key "vpn.server.daemon.tcp.port" --value "443" ConfigPut
./sacli start
```

As an aside, it is not possible to have the OpenVPN UDP daemons and OpenVPN TCP daemons listen on 2 separate interfaces, they have to listen on the same interface. If you do want to change this you can use iptables to redirect traffic on a specific port and interface internally to the correct port and interface.

Disable multi-daemon mode and use only TCP or UDP

Because the OpenVPN 2 code base is single-thread, meaning that an OpenVPN process can run on only 1 CPU core and doesn't know how to make use of multi-core systems, the OpenVPN Access Server comes with the ability to launch multiple OpenVPN daemons at the same time. Ideally there would be one OpenVPN daemon for every CPU core. But there's more involved. To make it possible for OpenVPN clients to establish a connection via the UDP protocol and via the TCP protocol, there are additional OpenVPN daemons necessary. In the case of the OpenVPN Access Server this means we launch 1 TCP and 1 UDP daemon per CPU core. On a system with 4 CPU cores this means there are in total 8 daemons running, 2 per CPU core; 1 TCP and 1 UDP. The Access Server performs a sort of internal load balancing. When connections come in, the Access Server decides which CPU core and thus which OpenVPN daemon is least busy, and connects you to that daemon.

In some rare cases it can be desirable or necessary to turn off multi-daemon mode and simply launch one TCP or UDP OpenVPN daemon, and handle all incoming OpenVPN tunnel connections through one single OpenVPN daemon. This is possible but has some possibly negative side effects. For one, service forwarding is used to field incoming browser requests on the TCP OpenVPN daemons on port TCP 443, and internally redirect them to the actual web services port TCP 943 internally. Disabling the TCP OpenVPN daemons and running with only one UDP daemon means normal access to the web services has now been blocked and you have to manually type the correct port number in the URL like so: **https://vpn.yourserver.com:943/**. Another side effect is that on restrictive networks where UDP connections are blocked, but TCP 443 (the default HTTPS port) is still open, then while running only an UDP OpenVPN daemon you could be unable to make a connection from such a restrictive network, whereas with default settings it would likely

have worked. And if you decide to use TCP daemons only, then the TCP Meltdown phenomenon may adversely affect your connection. So in short; the defaults are the best, but if you want to, you can disable multi-daemon mode.

To disable multi-daemon mode and use only 1 TCP daemon:

```
./sacli --key "vpn.server.daemon.enable" --value "false" ConfigPut
./sacli --key "vpn.daemon.0.listen.protocol" --value "tcp" ConfigPut
./sacli --key "vpn.server.port_share.enable" --value "true" ConfigPut
./sacli start
```

To disable multi-daemon mode and use only 1 UDP daemon:

```
./sacli --key "vpn.server.daemon.enable" --value "false" ConfigPut
./sacli --key "vpn.daemon.0.listen.protocol" --value "udp" ConfigPut
./sacli --key "vpn.server.port_share.enable" --value "false" ConfigPut
./sacli start
```

Reset multi-daemon mode and number of TCP/UDP daemons

In the commands below we are using the **sacli GetNCores** command to get the amount of CPU cores detected on this system, and then use that to configure the amount of TCP daemons and amount of UDP daemons to spawn when Access Server starts. The characters around the **./sacli GetNCores** command in the commands shown below are backticks , not single quotes, and this makes a significant difference in how the command is executed. We recommend copying and pasting the commands to be sure the commands are executed properly. We are also resetting the default setting here to use multi-daemon mode where multiple OpenVPN daemons are launched.

Restore the default of using multi-daemon mode, with the amount of processes same as CPU cores (recommended):

```
./sacli --key "vpn.server.daemon.enable" --value "true" ConfigPut
./sacli --key "vpn.daemon.0.listen.protocol" --value "tcp" ConfigPut
./sacli --key "vpn.server.port_share.enable" --value "true" ConfigPut
./sacli --key "vpn.server.daemon.tcp.n_daemons" --value "`./sacli GetNCores`"
ConfigPut
./sacli --key "vpn.server.daemon.udp.n_daemons" --value "`./sacli GetNCores`"
ConfigPut
./sacli start
```

Reset OpenVPN web services and daemons to defaults

Aside from offering you the chance to undo any wrong settings that have accidentally locked out of access to the web services, these steps are also vital when you have restored a backup of an OpenVPN Access Server configuration from one system to another system, and the interface names that were on the old server are not the same as the new server. If for example on the old server you have it configured to listen only to eth0, but the new server only has ens192, then you have a problem since the Access Server will be unreachable and you can't access the Admin UI to correct these settings. With the

commands below these settings related to interface names and such are all reset to "all", meaning that the OpenVPN Access Server will simply listen to all available interfaces, and at the default ports (TCP 443, TCP 943, UDP 1194).

Reset web services, service forwarding, and OpenVPN daemons to default ports and listen on all interfaces:

```
./sacli --key "admin_ui.https.ip_address" --value "all" ConfigPut
./sacli --key "admin_ui.https.port" --value "943" ConfigPut
./sacli --key "cs.https.ip_address" --value "all" ConfigPut
./sacli --key "cs.https.port" --value "943" ConfigPut
./sacli --key "vpn.server.port_share.enable" --value "true" ConfigPut
./sacli --key "vpn.server.port_share.service" --value "admin+client" ConfigPut
./sacli --key "vpn.daemon.0.server.ip_address" --value "all" ConfigPut
./sacli --key "vpn.daemon.0.listen.ip_address" --value "all" ConfigPut
./sacli --key "vpn.server.daemon.udp.port" --value "1194" ConfigPut
./sacli --key "vpn.server.daemon.tcp.port" --value "443" ConfigPut
./sacli start
```

You can also instead choose to specify IP address to listen on instead of "all", in case you want to set this manually.

XML-RPC interface

The OpenVPN Access Server uses XML-RPC internally between web services and core component, and between OpenVPN Connect Client for Windows and Macintosh, and the XML-RPC interface on the web services (at /RPC2 URL). On the OpenVPN Connect Client it is only used in a limited fashion to check credentials to see if they are valid, and to obtain a user-locked profile for connecting, when the Connect Client is using a server-locked profile. If the XML-RPC interface setting is changed to full support, either in the **Client Settings** page in the Admin UI, or via the command line with the configuration option shown below, then the Access Server can be fully remotely controlled using XML-RPC calls instead. Authentication is done via HTTP basic authentication over a secure SSL connection. To retrieve a user-locked profile a standard user's credentials are sufficient, but for other functions only an admin user's credentials are sufficient.

We do not provide documentation or support for the XML-RPC interface.

However, we can give you the tools to determine what calls to make and how, and you can use that information to use or make XML-RPC capable programs that can remotely control the Access Server.

To see XML-RPC calls on the command line with the **sacli VPNSummary** function:

```
OPENVPN_AS_DEBUG_XML=1 ./sacli VPNSummary
```

You will get a result which shows the XML query, and the response. This system of getting information works for pretty much every sacli function. And sacli controls just about everything that the Access Server can do.

To change the XML-RPC function support:

```
./sacli --key "xmlrpc.relay_level" --value <NUMBER> ConfigPut
./sacli start
```

Where <NUMBER> is:

- **0** - disable the XML-RPC API via web services entirely, and will break server-locked profile type connections.
- **1** - enable XML-RPC API via web services in a limited fashion, for server-locked profile type connections only (default).
- **2** - fully enable XML-RPC API via web services, allows full remote control of the Access Server's function.

Logging of XML-RPC API calls is by default not enabled, but can be enabled with an XML-RPC debug flag.

Limit total maximum amount of VPN tunnels

By default the Access Server allows 2048 VPN tunnels on a single installation of Access Server. This is normally enough, but if you want to, you can increase that limit. Please note that if you change this value, even a warm restart of Access Server will restart the OpenVPN daemons, meaning all your VPN clients get kicked off and they will need to reestablish their connection, which should happen automatically.

Change maximum amount of active incoming VPN tunnels:

```
./sacli --key "vpn.server.max_clients" --value <NUMBER> ConfigPut
./sacli start
```

Where <NUMBER> is the maximum amount of connected VPN tunnels. This configuration key by default is not present in Access Server, and when it is not present, it will be assumed to be 2048. It can be set to any valid number of your choice.

UCARP/VRRP failover advanced settings

When the built-in failover mode of the Access Server is configured and in use, the primary node will send out heartbeat signals onto the local network. The secondary node monitors these heartbeat signals, and if it fails, takes over the tasks from the failed node. But if multiple such pairs are active on the same network, or if other systems also use UCARP/VRRP for automatic failover, then the system needs a way to differentiate the signals. This is done with a VHID which is a unique number embedded in the heartbeat signals. Each failover pair needs its own ID. By default this number is 94 on an Access Server failover pair. To adjust it to another number adjust the value of the **ucarp.vhid** configuration key with the command below, but beware that you should follow the steps carefully as described below for both nodes, and that this will lead to having to restart the Access Server service on each node in turn, causing a total of 2 failover events. So plan this appropriately.

On the primary node adjust the VHID:

```
./sacli --key "ucarp.vhid" --value <NUMBER> ConfigPut
service openvpnas restart
```

Where is a number from 1 to 255.

Now wait a full minute. This is to ensure that the primary node has had a chance to create a new configuration backup file and to relay it to the secondary node. There will now be a brief moment where both nodes try to be the master node, as each does not see the other anymore due to the mismatched VHID number.

Now go to the secondary node and restart the Access Server service:

```
service openvpnas restart
```

The primary node should now come back online properly and the secondary node should now be in standby mode again.

Finally, for advanced users, it is possible to pass additional parameters to the UCARP process. This is done with the **ucarp.extra_parms** configuration key. See the command below on how to pass extra parameters to the UCARP process that Access Server manages. Please note that changing this will result in a failover event and you will then have to restart the Access Server service on the secondary node as well to ensure it goes back the primary node.

Define extra parameters for Access Server to pass to UCARP:

```
./sacli --key "ucarp.extra_parms" --value <PARAMETERS> ConfigPut
service openvpnas restart
```

Where is a string of text that contains what you want to pass to UCARP.

If for example you want to override the standard scripts that OpenVPN Access Server uses for when the node becomes active or has to be a standby node, then you can do so by passing new **--upscript** and --**downscript** parameters directly to UCARP, and specifying new scripts instead. You could for example copy the original **ucarp_standby** and **ucarp_active** up/down scripts in the **/usr/local/openvpn_as/scripts/** directory and edit them to suit your needs. It is of course possible to edit the scripts directly but that would mean during an upgrade or reinstallation that these scripts are reset to standard. Using the method described to create your own copies of the up/down scripts that you can customize is the better method if you want to customize these up/down scripts.

Override up/down scripts with new scripts (make sure to create them of course):

```
./sacli --key "ucarp.extra_parms" --value "--upscript /root/up --downscript
/root/down" ConfigPut
service openvpnas restart
```

And to revert to the default scripts:

```
./sacli --key "ucarp.extra_parms" ConfigDel
service openvpnas restart
```

Global NAT behavior setting

Since private IP addresses cannot be routed on the Internet, when VPN clients are connected to the Access Server and have been given instructions to send traffic for public IP addresses through the VPN server, the Access Server will choose the network interface with the default gateway on it and NAT traffic out through there. In some cases it is desirable to disable this NAT behavior, for example when you wish to implement a firewall system that logs the VPN clients private IP addresses as the traffic passes from the VPN client, through the VPN server, through the firewall, and then goes to the Internet. The NAT behavior can then be implemented further on in the connection chain before it goes onto the public Internet. This is a global setting that applies to the entire server for outgoing traffic through NAT. It is possible to disable this setting, or to specify a different IP address to use for outgoing NAT, or even a range of addresses that will be randomly selected for outgoing NAT operations. It is impossible to bind a specific public IP for outgoing NAT operations to a specific VPN client.

Disable NAT for outgoing public traffic (enabled by default):

```
./sacli --key "vpn.server.nat" --value "false" ConfigPut
./sacli start
```

Re-enable NAT (restore default):

```
./sacli --key "vpn.server.nat" ConfigDel
./sacli start
```

Specify interface/address for outgoing NAT:

```
./sacli --key "vpn.server.routing.snat_source.N" <INTERFACE-ADDRESS>
./sacli start
```

Where **N** is a number starting from **0** and logically increments, for multiple definitions. And where **INTERFACE-ADDRESS** is one of the following:

- **interface:address** - Source NAT traffic using IP address of a specified interface name.
- **interface:number** - Source NAT using IP address of alias number of specified interface name.
- **interface:begin-range:end-range** - Source NAT traffic at random using range of IP addresses.

The randomization of that last option is done using the Linux/Netfilter to-source algorithm. It is of course required that the interfaces and IP addresses you intend to use are actually available and configured on your system and are by themselves working properly. Examples of specifying the interface and address for outgoing NAT are given below.

For example NAT eth2 traffic via 1.2.3.4:

```
./sacli --key "vpn.server.routing.snat_source.0" --value "eth2:1.2.3.4" ConfigPut
./sacli start
```

Or NAT eth0 traffic via the eth0:4 address:

```
./sacli --key "vpn.server.routing.snat_source.0" --value "eth0:4" ConfigPut
./sacli start
```

Or NAT ens192 traffic using a range of public IPs from 76.49.27.18 to 76.49.27.22:

```
./sacli --key "vpn.server.routing.snat_source.0" --value
"ens192:76.49.27.18:76.49.27.22" ConfigPut
./sacli start
```

Multiple rules can be specified for multiple interfaces, for example:

```
./sacli --key "vpn.server.routing.snat_source.0" --value
"eth0:76.49.27.18:76.49.27.22" ConfigPut
./sacli --key "vpn.server.routing.snat_source.1" --value "eth1:3" ConfigPut
./sacli start
```

Settings related to iptables

The Access Server makes heavy use of Linux iptables to enable NAT functionality and enforce VPN-level access control rules, however it also tries to play well with other applications that use iptables by maintaining its own chains and making minimal additions to standard chains such as INPUT, OUTPUT, and FORWARD. By default, the Access Server prepends to standard chains, and this remains the default. Prepending means it tries to come first in an existing list of iptables settings, to ensure Access Server works properly. However by using the following config key, this behavior can be changed to append, to make it easier to develop custom rules which take priority over Access Server-generated rules.

To make Access Server add rules after existing ones (append instead of prepend):

```
./sacli --key "iptables.append" --value "True" ConfigPut
./sacli start
```

Restore default behavior:

```
./sacli --key "iptables.append" ConfigDel
./sacli start
```

It is also possible to completely disable Access Server's activities in regards to iptables. However, this may lead to insecure situations as traffic may be allowed through that you didn't give permission for, and things may then simply not function as intended anymore. Disabling iptables means you're taking away one of the pillars on which the Access Server functionality is based and you are then expected to take care of the required actions in iptables yourself. If you do not, the Access Server will likely just completely fail to function. We do not recommend disabling Access Server managing the iptables settings. But if you must, for whatever reason, and you have the required knowledge to get things working, then the option is available. There are 3 distinct iptables items that Access Server manages and these that are all enabled by default, but can optionally be disabled:

- **iptables.vpn.disable.filter**
- **iptables.vpn.disable.nat**
- **iptables.vpn.disable.mangle**

Example for disabling one of the three above settings:

```
./sacli --key "iptables.vpn.disable.filter" --value "True" ConfigPut
./sacli start
```

Restoring the value to its default:

```
./sacli --key "iptables.vpn.disable.filter" ConfigDel
./sacli start
```

Choosing Layer 3 (routing) or Layer 2 (bridging)

Before we explain this setting further, we want to make it clear that layer 2 VPN mode or bridging is not a recommended method of using OpenVPN Access Server, and you may encounter problems with it that are related to MAC address spoofing or promiscuous mode which are security related settings in hardware and software that may need to be adjusted or enabled in order for this to work at all. We also want to warn you that using bridging mode disables a lot of the functionality of the Access Server because it simply does not apply anymore. In layer 3 mode, the recommended mode, the Access Server functions as a router with firewall functions built-in to ensure traffic can't go to places it shouldn't be able to go. But with layer 2, you're basically turning the Access Server into a software-based network switch with encryption where all connected VPN clients can communicate freely with each other and the network the Access Server is attached to. There's no control here over what traffic is allowed to go where, and the Access Server also plays no role in assigning IP addresses or specific access rules to the VPN clients. In other words, if you don't know what you're doing, do not use this mode and stick to the default Layer 3 routing mode, please.

To learn about what Layer 3 and Layer 2 means see ourshort explanation on what the OSI Layer model is.

The configuration parameter **vpn.general.osi_layer** controls the behavior of the Access Server. It can operate on Layer 2 in bridging mode, and in Layer 3 in routing mode (the default). In the past, in Access Server versions older than version 2.5, it was possible to set this option in the Admin UI, but we have since hidden this option further to prevent people from trying it out accidentally, as it is a very advanced feature and likely to cause the product to appear not to function anymore, unless you know what you're doing. But the option for Layer 2 bridging mode can still be enabled.

Switch to Layer 2 bridging mode:

```
./sacli --key "vpn.general.osi_layer" --value "2" ConfigPut
./sacli start
```

Restore to Layer 3 routing mode:

```
./sacli -key "vpn.general.osi_layer" ConfigDel
./sacli start
```

If you had Access Server installed and operating on Layer 2 bridging mode already, and you have just upgraded your Access Server to the latest version, this setting will remain

intact and your server will continue to function in Layer 2 bridging mode. So an upgrade will not break this functionality.

If you had all your VPN clients installed and operating on Layer 3 routing mode already, and you now switch your server to Layer 2 bridging mode, any VPN clients that have a stored copy of the user-locked or auto-login type connection profiles will need to obtain a new copy of the connection profile before they are able to successfully connect again.

The most common problems we encounter with Layer 2 are that the VPN client does not get an IP address assigned. The most common reason for this is that you now need a DHCP server running either on the Access Server itself or on the network that the Access Server is connected to (but not both at the same time), and that either such a DHCP server does not exist, or is unreachable because the network or the device that the DHCP server runs on has a security feature that is called MAC address spoofing or promiscuous mode set to a safe level. These terms both describe the same idea, where a single computer, in this case the Access Server, pretends to be multiple systems at the same time, which makes sense in this case, because it tries to handle traffic for multiple VPN clients that all want connectivity to the connected network. On virtual platforms like ESXi or HyperV you may need to look into these settings on the virtual switch and allow this type of behavior on the network before Layer 2 bridging mode can function.

Please note that due to the added complexity of implementing Layer 2 bridging mode where external equipment is usually the cause of the problem, we may not be able to offer you adequate support in resolving this issue. We can only ensure the Access Server and the OpenVPN clients can make a connection, but IP addressing and traffic transmission issues that pass the boundary where Access Server connects to your network, and doesn't function from there on, is not something we can resolve from our end.

Allow UDP multicast and IGMP to pass through

By default in Layer 3 routed mode, which is what the Access Server uses normally, all traffic is unicast. That means that only traffic that has a specific destination IP address will be allowed to pass through the VPN server. Multicast traffic, or broadcast traffic that has a to-whom-it-may-concern characteristic, is blocked. It is possible to lift the restriction on UDP multicast packets and IGMP packets, so that these pass freely between VPN clients and the VPN server. Some software programs use these to auto-detect systems or services on the network, and so this option may be useful in such a situation.

The configuration key **vpn.routing.allow_mcast** allows this traffic to pass through. It is disabled by default.

Enable UDP multicast and IGMP traffic passthrough:

```
./sacli --key "vpn.routing.allow_mcast" --value "true" ConfigPut
./sacli start
```

Restore the default setting:

```
./sacli --key "vpn.routing.allow_mcast" ConfigDel
./sacli start
```

This setting implements these iptables rules on the VPN server, which is what allows the traffic to pass through:

```
ACCEPT udp   -- anywhere  base-address.mcast.net/4 udp
ACCEPT igmp  -- anywhere  anywhere
ACCEPT udp   -- anywhere  base-address.mcast.net/4 udp
ACCEPT igmp  -- anywhere  anywhere
```