Secure,

Multi-lateral Peering

with

OpenSER 1.0.0

Revision History

Revision	Date of Issue	Changes
0.1	12 Aug 2005	Initial Draft.
0.2	23 Aug 2005	Minor clarifications and edits
0.3	4 Nov 2005	Added Appendix 1 and other edits
0.4	15 Nov 2005	Edited for OpenSER V1.0.0

E-mail: support@transnexus.com
www.transnexus.com

Contents

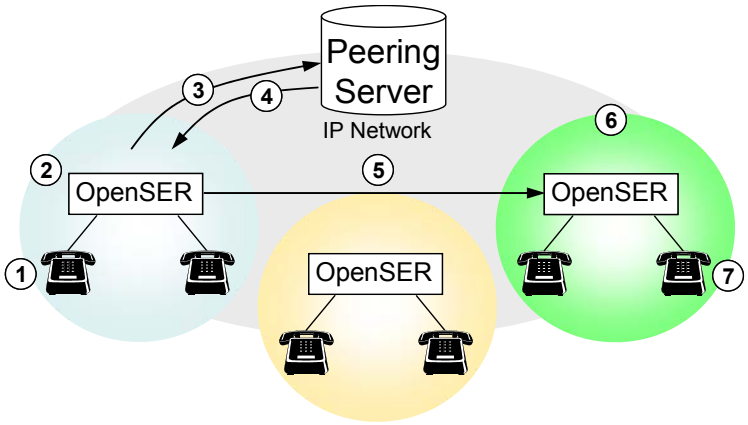
Introduction

Secure multi-lateral peering uses Public Key Infrastructure (PKI) services to secure, direct peering among an anonymous group of SIP peers. In a multi-lateral peering architecture, each peer trusts a common peering authority that enforces routing and access policies on behalf of each peer. The benefits of multi-lateral peering are increased peering security and the elimination of burdensome bilateral peering agreements and access control lists which are difficult to administer in a large peering network.

This document provides build instructions on how to build the OSP Toolkit with OpenSER. The OSP Toolkit, which is freely available from www.sipfoundry.org, contains an implementation of the OSP standard defined by the European Telecommunications Standards Institute (ETSI TS 101 321) www.esti.org. The OSP Toolkit enables the SIP Express Router for secure multi-lateral peering.

Multi-lateral SIP Peering

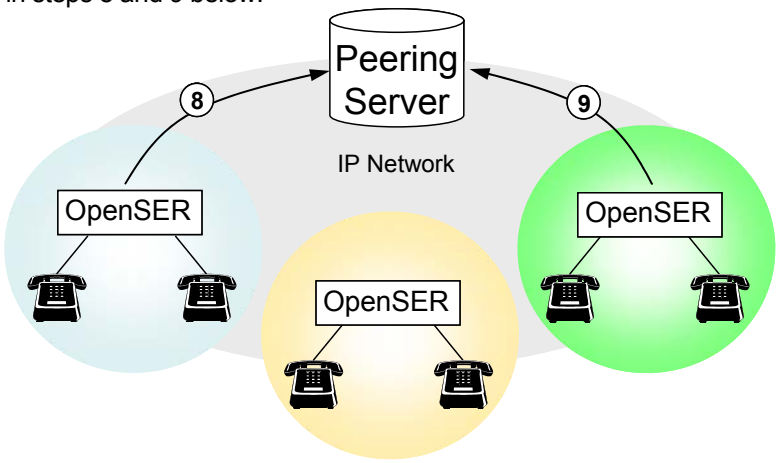
A peering server is a simple and efficient solution for managing routing, access control and CDR collection for VoIP calls among a network of OpenSER devices. OSP can be used to securely manage wholesale VoIP peering among independent SIP networks, or by an enterprise to create a secure VoIP virtual private network for calling among branch offices using SIP PBXs. The diagram below illustrates a call scenario between OpenSER networks using OSP peering. Each OpenSER proxy manages calls within its own domain. However, when a call must be completed outside its own network, an OpenSER proxy can query a peering server for routing and access information to a destination peer that can complete the call.



1. The calling party makes a call.
2. The source OpenSER cannot complete the call within its domain.
3. Peering Request. The source OpenSER queries the peering server for the IP addresses of other peers that can complete the call to the dialed number.
4. Peering Response. The peering server returns a list of IP addresses of destination peers and digitally signed peering tokens authorizing access to each destination peer.
5. The source OpenSER routes the call to the destination OpenSER returned by the peering server. Included in the SIP Invite message is the peering access token signed by the peering server.
6. The destination OpenSER receives the call and validates the peering token. If the token is valid, the destination OpenSER routes the call to the called telephone number.
7. The call is completed to calling party.

Call Detail Record Collection

When the call is over, both the source and destination peers send call detail records to the peering server as shown in steps 8 and 9 below.



Step 1. Build the OSP Toolkit

The OSP Toolkit, when compiled, is a library comprised of OSP client functions that simplify sending and receiving OSP peering messages. It is this library, which will be integrated into the SER. The OSP Toolkit uses third party software (by default OpenSSL) for cryptographic algorithms and for secure internet transactions (HTTPS). The OSP Toolkit also includes the application *Enroll* which enables the OSP client device to generate its own public-private key pair, get the public key from an OSP peering server, send a certificate request to a peering server and receive the resulting signed certificate from the peering server.

In order to successfully compile and use the OSP Toolkit, the following list of software is required:

OpenSSL (required) - Open Source SSL protocol and Cryptographic Algorithms (version 0.9.7g recommended) from www.openssl.org. Pre-compiled OpenSSL packages are not recommended because of the binary compatibility issue.

Perl (required) - A programming language used by OpenSSL for compilation. Any version of Perl will work. One version of Perl is available from www.activestate.com/ActivePerl. If pre-compiled OpenSSL packages are used, Perl package is not required.

C compiler (required) - Any C compiler should work. The GNU Compiler Collection from www.gnu.org is routinely used for building the OSP Toolkit for testing.

OSP Server (required for testing) - Access to any OSP server should work. OpenOSP is an open source server available from www.sipfoundry.org. Also, osptestserver.transnexus.com is freely available for testing – contact support@transnexus.com for testing access.

Unpacking the Toolkit

After downloading the OSP Toolkit from www.sipfoundry.org, perform the following steps in order:

1) Copy the OSP Toolkit distribution into the directory where it will reside.

2) Unzip the tar file by executing the following command:

```
gunzip OSPToolkit-###.tar.gz
```

Where ### is the version number separated by underlines. For example, if the version is 2.5.4, then the above command would be:

`gunzip OSPToolkit-2_5_4.tar.gz`

A tar file will replace the *.gz file.

3) Untar the file by executing the following command:

```
tar -xvof OSPToolkit-###.tar
```

Where ### is the version number separated by underlines. For example, if the version is 2.5.4, then the above command would be: `tar -xvof OSPToolkit-2_5_4.tar`

A new directory (`osptoolkit`) will be created within the same directory as the tar file.

4) Go to the osptoolkit directory by running this command:

```
cd osptoolkit
```

Within this directory, you will find directories and files similar to what is listed below (if the command "`ls -F`" is executed):

```
> ls -F
enroll/
RelNotes.txt                    lib/
README.txt                      license.txt
bin/                            src/
crypto/                         test/
include/
```

Preparing to Build the OSP Toolkit

5) Compile OpenSSL according to the instructions provided with the OpenSSL distribution. **You would need to do this only if you don't have openssl already.**

6) Copy the OpenSSL header files (ones with the *.h extension contained in the `include/openssl` directory within the directory containing OpenSSL files into the `crypto/openssl` directory within the Toolkit. The OpenSSL header files are located under the `include/openssl` directory.

7) Copy the OpenSSL library files (`libcrypto.a` and `libssl.a`) into the lib directory within the Toolkit. The OpenSSL library files are located within the `openssl` directory.

Building the OSP Toolkit

8) Open the makefile in the `src` directory. Look for the install path variable – INSTALL_PATH; edit it to be `/usr/local`.

```
cd src; vi Makefile
```

9) From within the OSP Toolkit directory, start the compilation script by executing the following commands:

```
cd src

make clean; make build
```

The make script can also be used to install the toolkit header files and the toolkit library in the folder that the user wishes to install them in. The default path is: `/usr/local/`. The user can modify this to any path desired. Also, the "`make install`" command should be used to install the files.

Note: Please make sure you have the rights to access the INSTALL_PATH directory. For example, In order to access `/usr/local` directory, normally, you should be root.

Building the Enroll Utility

The Enroll program is a utility application for establishing a trusted relationship between the SER OSP client and an OSP peering server or certificate authority. This step will build the Enroll utility included with the OSP Toolkit.

10) From within the OSP Toolkit directory, execute the following commands at the Unix command prompt:

```
cd enroll

make clean; make linux
```

Compilation is successful if there are no errors anywhere in the compiler output. The enroll program is now located in the OSP Toolkit "bin" directory. For more information on the Enroll utility, please see the document "Device Enrollment" at www.sipfoundry.org/OSP/OSPclient.

Step 2. Enroll OpenSER with a Peering Server

Overview

To establish a secure relationship between an OSP peering server and the OSP Module in OpenSER requires three crypto files. These files are:

- `localcert.pem` - The local certificate for SER signed by the OSP server.
- `pkey.pem` - The private key generated by the Enroll utility for SER.
- `cacert_#.pem` - The Certificate Authority (CA) certificate from an OSP server. OpenSER may enroll with multiple certificate authorities or peering servers. The # represents an integer indicating the CA certificate from different peering servers.

The Enroll utility automates the process of enrolling SER with a peering server and creating the three crypto files. By default, the OSP Module of OpenSER will load the crypto files from the default configuration directory – `/usr/local/etc/openser`. If the files are not present, the OSP Module and OpenSER will not start.

Using the Enroll script

The script '`enroll.sh`' requires AT&T korn shell (ksh) or any of its compatible variants. The `enroll.sh` script should be run from the `osptoolkit/bin` directory, or the `osptoolkit/bin` directory should be in the PATH variable.

From the command line, type `enroll.sh` followed by the IP address or domain name of the peering server. Below is an example of the Enroll utility being used to enroll OpenSER with a peering server named `osptestserver.transnexus.com`. The gray boxes indicate optional input which will be included in the OpenSER's certificate. Error Code 0 indicates the operation was successful with no error.

```
# enroll.sh osptestserver.transnexus.com
Generating a 512 bit RSA private key
..........................++++++++++++
.++++++++++++
writing new private key to 'pkey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

Error Code returned from openssl command : 0

CA certificate received
[SP: osptestserver.transnexus.com]Error Code returned from getcacert command : 0

output buffer after operation: operation=request
output buffer after nonce: operation=request&nonce=6096834216798074
X509 CertInfo context is null pointer
Unable to get Local Certificate
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify error:num=18:self signed certificate
verify return:1
depth=0 /CN=osptestserver.transnexus.com/O=OSPServer
verify return:1
The certificate request was successful.
Error Code returned from localcert command : 0
```

Note: The `localcert.pem`, `pkey.pem` and `cacert_#.pem` files generated by the Enroll utility must be copied to the `/usr/local/etc/openser` directory.

Step 3. Compile OpenSER with OSP

Obtain the OpenSER code

Download OpenSER 1.0.0 from http://openser.org.

```
gunzip openser-1.0.0_src.tar.gz

tar –xvf openser-1.0.0_src.tar
```

It will create a sub-folder starting with openser-1.0.0. The distribution comes with a variety of modules (under "modules" folder), but does not include the OSP peering module yet. So, download the module separately and unpack it under openser-1.0.0/modules folder.

Obtain the OSP peering module

Download OSP peering module 0.2.2 from http://osp-module.berlios.de and follow the links to Project and Files, or directly from http://download.berlios.de/osp-module/osp-0_2_2.tar.gz Un-package (unzip/untar) the file under openser-1.0.0/modules directory. It should create a new folder "osp"

```
cd openser-1.0.0/modules; gunzip osp-0_2_2.tar.gz

tar –xvf osp-0_2_2.tar
```

Compile OpenSER with the OSP module

Build OpenSER with the OSP peering module. Log in as root, go to the openser-1.0.0 folder and run "`make install`". It will build the OpenSER with all modules (including OSP), and distribute executable and configuration files around the system.

```
cd ..; make install
```

Step 4. Configure OpenSER

`openser.cfg` is the main configuration file for OpenSER. It includes instructions for configuring OpenSER, loading external modules, configuring modules and CPL (call processing language) instructions. By default, the OpenSER looks for the configuration file in `/usr/local/etc/openser`. OpenSER includes a sample `openser.cfg` file, but it does not include instructions for using the OSP peering module. Instead of using the default `openser.cfg`, use the sample configuration file `modules/osp/etc/sample-osp-openser.cfg`. Copy this file to overwrite the default configuration - `/usr/local/etc/openser/openser.cfg`.

```
cp modules/osp/etc/sample-osp-openser.cfg /usr/local/etc/openser/openser.cfg
```

The only parameter that must be configured in the `openser.cfg` is the location of a peering server. Edit the file and update the value of "`sp1_uri`" parameter to indicate the URL of the peering server. Configuring the "`sp2_uri`" parameter to load share and add redundancy with a second peering server is recommended. Two other parameters which are recommended for editing are "`device_ip`" and "`token_format`". More details on these parameters are provided in the `sample-osp-openser.cfg` file. Appendix 1 provides detail on all OSP peering module parameters.

An additional tuning adjustment for optimizing OpenSER is to decrease the number of TCP synchronization retries made by the Linux operating system when a TCP connection fails. Setting this parameter to 0 or 1 will decrease post dial delay when a TCP connection to a destination peer is not possible. Reducing TCP synchronization retries by the operating system will speed call fail-over to the next destination peer. To change this parameter, open the file `/etc/sysctl.conf` and add the entry `net.ipv4.tcp_syn_retries = 1`. Reboot the computer and check the parameter entry was accepted by typing `cat /proc/sys/net/ipv4/tcp_syn_retries`. It should display "1".

Step 5. Run OpenSER

Running OpenSER requires root privileges. To start the OpenSER, use the following command

```
openser-1.0.0/rpm/openser.init start
```

To stop the OpenSER, use the following command

```
openser-1.0.0/rpm/openser.init stop
```

Frequently Asked Questions

This section includes common questions about the OSP peering module with OpenSER along with answers. If your question was not answered here, please submit your question to the OSP mailing list at www.sipfoundry.org or e-mail support@transnexus.com.

Where are OSP peering module log messages written?

Log messages are sent to the sys-log application and will be logged according to the logger's configuration. Look for the messages in the `/var/log/messages` file.

Can multiple instances of OpenSER be run on the same machine?

It is possible to run more than one OpenSER on the same computer. The simplest way is to create a second `openser.cfg` and edit the port number from 5060 to a different port number. When starting the second OpenSER, instead of using the `rpm/openser.init` script, explicitly specify where the configuration file is using the `-f config-file.openser` flag. If the second OpenSER requires a

different set of certificates, uncomment and update the "`private_key`", "`local_certificate`" and "`ca_certificate`" variables in the configuration file.

Can I use both TCP and UDP for communication?

OpenSER supports both UDP and TCP. A peering response does not explicitly indicate which transport protocol to use for communicating to the terminating SIP proxy or user agent. By default, OpenSER uses to UDP. Insure that the terminating proxy or UA is defined on the peering server as a SIP device. This will instruct the peering server to use a compressed peering token, small enough to fit into a single UDP packet.

How is OSP authentication different from password or access list authentication?

The OSP peering module implements Public Key Infrastructure (PKI) services to securely authenticate and authorize peer to peer calling among SIP proxies. If the SIP INVITE received by OpenSER includes a peering authorization token, the OSP peering module will validate the peering token was digitally signed by the OpenSER's peering authority. If the token is valid the OpenSER will accept the call. If not, OpenSER will reject the call.

The OSP peering module uses the public key of its peering authority to verify that its peering authority signed the peering token (standard asymmetric cryptography). When compared to access lists or passwords for peer to peer authentication and authorization, PKI services offer improved security and scalability. Passwords, shared secrets or symmetric keys are prone to security breaches. Access lists have limited scalability. Managing an access list that contains hundreds of addresses and changes frequently is a major operations challenge. Secure peering using PKI services eliminates these limitations.

Why is time synchronization with the OSP peering server important?

Peering authorization tokens have a limited life – usually five minutes. If the time of day of OpenSER differs by more than five minutes from its peering server, peering tokens from SIP INVITES received from other peers will be invalid and calls will be rejected. The simple solution is for all peers to use Network Time Protocol (ntp) to maintain time synchronization.

Appendix 1 - OSP Peering Module Parameters

This Appendix provides a detailed explanation of all OSP peering module parameters that may be included in the `openser.cfg` file. Information on OSP peering module functions can be found in the "OSP Module for Secure, Multi-Lateral Peering" document located at: http://www.openser.org/docs/modules/1.1.x/osp.html.

sp1_uri, sp2_uri

These string parameters define peering servers to be used for requesting peering authorization and routing information. `sp1_uri` must be configured. `sp2_uri` is required only if there are two peering servers. Each peering server address takes the form of a standard URL, and consists of up to four components:

- An optional indication of the protocol to be used for communicating with the peering server. Both HTTP and HTTP secured with SSL/TLS are supported and are indicated by "`http://`" and "`https://`" respectively. If the protocol is not explicitly indicated, the OpenSER defaults to HTTP secured with SSL.
- The Internet domain name for the peering server. An IP address may also be used, provided it is enclosed in square brackets such as [172.16.1.1].
- An optional TCP port number for communicating with the peering server. If the port number is omitted, the OpenSER defaults to port 1080 (for HTTP) or port 1443 (for HTTP secured with SSL).
- The uniform resource identifier for requests to the peering server. This component is not optional and must be included.

Examples:

```
modparam("osp", "sp1_uri", "http://osptestserver.transnexus.com:1080/osp")
modparam("osp", "sp2_uri", "https://[1.2.3.4]:1443/osp")
```

device_ip

`device_ip` (string) is a recommended parameter that explicitly defines the IP address of OpenSER in an peering request message (as SourceAlternate type=transport). The IP address must be in brackets as shown in the example below.

Example:

```
modparam ("osp", "device_ip", "[1.1.1.1]")
```

token_format

When OpenSER receives a SIP INVITE with a peering token, the OSP peering module will validate the token to determine whether or not the call has been authorized by a peering server. Peering tokens may, or may not, be digitally signed. The `token format` (integer) parameter defines if OpenSER will validate signed or unsigned tokens or both. The values for `token format` are defined below. The default value is 2.

0 - Validate only signed tokens. Calls with valid signed tokens are allowed.
1 - Validate only unsigned tokens. Calls with valid unsigned tokens are allowed.
2 - Validate both signed and unsigned tokens are allowed. Calls with valid tokens are allowed.

Example:

```
modparam ("osp", "token_format", 2)
```

private_key
local_certificate
ca_certificates

These parameters identify crypto files are used for validating peering authorization tokens and establishing a secure channel between OpenSER and a peering server using SSL. The files are generated using the 'Enroll' utility from the OSP Toolkit. By default, the proxy will look for `pkey.pem`, `localcert.pem`, and `cacart_0.pem` in the default configuration directory. The default config directory is set at compile time using CFG_DIR and defaults to `/usr/local/etc/ser/`. The files may be copied to the expected file location or the parameters below may be changed.

Example: If the default CFG_DIR value was used at compile time, the files will be loaded from:

```
modparam("osp", "private_key",      "/usr/local/etc/openser/pkey.pem")
modparam("osp", "local_certificate","/usr/local/etc/openser/localcert.pem")
modparam("osp", "ca_certificates",  "/usr/local/etc/openser/cacert_0.pem")
```

sp1_weight, sp2_weight

These `sp_weight` (integer) parameters are used for load balancing peering requests to peering servers. These parameters are most effective when configured as factors of 1000. For example, if `sp1_uri` should manage twice the traffic load of `sp2_uri`, then set `sp1_weight` to 2000 and `sp2_weight` to 1000. Shared load balancing between peering servers is recommended. However, peering servers can be configured as primary and backup by assigning a `sp_weight` of 0 to the primary server and a non-zero `sp_weight` to the back-up server. The default values for `sp1_weight` and `sp2_weight` are 1000.

Example:

```
modparam ("osp", "sp1_weight", 1000)
```

device_port

The `device_port` (string) parameter is an optional field which includes the SIP port being used by OpenSER in the peering request (as SourceAlternate type=network) to the peering server. If is not configured, this information is not included in the peering request. This field is useful if multiple OpenSERs are running on the same Linux computer since it enables the peering server to administer different peering policies based on different SIP proxies.

Example:

```
modparam ("osp", "device_port", "5060")
```

enable_crypto_hardware_support

The `enable_crypto_hardware_support` (integer) parameter is used to set the cryptographic hardware acceleration engine in the openssl library. The default value is 0 (no crypto hardware is present). If crypto hardware is used, the value should be set to 1.

Example:

```
modparam ("osp", "enable_crypto_hardware_support", 0)
```

ssl_lifetime

The ssl_lifetime (integer) parameter defines the lifetime, in seconds, of a single SSL session key. Once this time limit is exceeded, the OSP peering module will negotiate a new session key. Communication exchanges in progress will not be interrupted when this time limit expires. This is an optional field with default value is 200 seconds.

Example:

```
modparam ("osp", "ssl_lifetime", 200)
```

persistence

The persistence (integer) parameter defines the time, in seconds, that an HTTP connection should be maintained after the completion of a communication exchange. The OSP peering module will maintain the connection for this time period in anticipation of future communication exchanges to the same peering server.

Example:

```
modparam ("osp", "persistence", 1000)
```

retry_delay

The retry_delay (integer) parameter defines the time, in seconds, between retrying connection attempts to an OSP peering server. After exhausting all peering servers the OSP peering module will delay for this amount of time before resuming connection attempts. This is an optional field with default value is 1 second.

Example:

```
modparam ("osp", "retry_delay", 1)
```

retry_limit

The retry_limit (integer) parameter defines the maximum number of retries for connection attempts to an peering server. If no connection is established after this many retry attempts to all peering servers, the OSP peering module will cease connection attempts and return appropriate error codes. This number does not count the initial connection attempt, so that a retry_limit of 1 will result in a total of two connection attempts to every peering server. The default value is 2.

Example:

```
modparam ("osp", "retry_limit", 2)
```

timeout

The timeout (integer) parameter defines the maximum time in milliseconds, to wait for a response from an peering server. If no response is received within this time, the current connection is aborted and the OSP peering module attempts to contact the next peering server. The default value is 10 seconds.

Example:

```
modparam ("osp", "timeout", 10)
```

max_destinations

The `max_destinations` (integer) parameter defines the maximum number of destinations that OpenSER requests the peering server to return in a peering response. The default value is 5.

Example:

```
modparam ("osp", "max_destinations", 5)
```

validate_call_id

The `validate_call_id` (integer) parameter instructs the OSP peering module to validate call id in the peering token. If this value is set to 1, the OSP module validates that the call id in the SIP INVITE message matches the call id in the peering token. If they do not match the INVITE is rejected. If this value is set to 0, the OSP module will not validate the call id in the peering token. The default value is 1

Example:

```
modparam ("osp", "validate_call_id", 1)
```