OpenVPN Gateway Builder

Introduction

Overview Installation Usage How it

A build system to create a minimalistic Linux system booting off a CD or USB stick for running OpenVPN VPN gateways or endpoints.

Introduction

OpenVPN is an open and operating-system agnostic VPN solution that has become very popular in the last years, especially with the recently published 2.0 release. With OpenVPN it is possible to create a secure VPN tunnel with as few as 5 lines in a configuration file, a feature that makes it very easy and convenient to use VPN tunneling as a solution to various network and connectivity problems. One major drawback to hardware solutions like a Cisco PIX is however the need to run a full operating system beneath an OpenVPN gateway. Even using Linux as OS, the administrative overhead of maintenance and patching remains high as opposed to a vendor hardware solution.

OpenVPN Gateway Builder (OGB) solves this problem by creating an embedded Linux system that can do exactly one job: Setup the network and run OpenVPN. There is no unneccessary software, that needs to be maintained, there is no filesystem on disk that could be compromised by an intruder (OGB runs only in RAM), there are no unneeded services installed etc. All binaries and kernel/modules for this Linux system are copied from the build machine, so that one has to first install the required software on the build machine before beeing able to use it in an OGB gateway.

Additionally OGB is very useful in the centralized administration of many OpenVPN gateways as all the configuration files and cryptographic material is stored only on the build machine, which can easily be secured. The CDs, which are used to boot the VPN gateways, are needed only to boot the gateway. After that they can be safely stored in a secure location. That way the cryptographic material is safe from spying or tampering.

The maintenance of the VPN gateway machines is reduced to maintaining the build machine using normal Linux maintenance mechanism, like RPM or debian updates. To distribute these updates to the VPN gateways one will simply re-create the boot media from the stored configuration and the updated versions of kernel, programs etc. will be used.

What is new ?

2007-08-17 OGB 1.4 released with minor fixes and improvements. I am still working on the new documentation, though.

2007-08-08 OGB 1.3 released with lots of serious improvements and bug fixes and new packages. Expect an update to the documentation here *real soon*

2006-04-07 OGB wins an award of the "Innovationspreis 2006 der Initiative Mittelstand" in the category of IT security. The award comes with a marketing bonus that goes to pro business Berlin AG, the company I work for.

2005-11-04 OGB 1.1 release with various smaller bugfixes and a new package for VMware.

License

OGB is licensed under the GNU General Public License

OpenVPN Gateway Builder

Introduction | Overview | Installation | Usage | How it wo

Minimalistic System

The basis of OGB is a very minimalistic Linux system, that is intended to run only a very limited amount of applications. To support that concept, OGB creates the Linux system from scratch out of the binaries and libraries found on the build system. Except for some hard-to-compile or seldom binaries that are included in the OGB distribution (like isolinux.bin), everything is taken from the build system.

Furthermore, the choice of included programs is limited to the absolute minimum required to run a Linux system. Even SSH is part of a package and can thus be left out if desired. As a result of this minimalistic approach there are no running daemons whatsoever (well, except syslog) and therefore no way of outside interference.

Copy binaries from Build System

Unlike many other mini-Linux system, OGB does not use µClib or BusyBox to reduce the required space or memory footprint. Instead, OGB copies the build systems' files and their dependancies which makes the OGB Linux system an exact copy of the build system regarding OS and program versions.

The OGB distribution does not contain any binary files (except isolinux.bin, which is strictly speaking not part of the running OGB system). This is so by design as OGB should be only a **build** system and not a Linux distribution of its own. It just helps one to create a mini linux out of your favourite Linux system.

This concept makes OGB a very lean base for various purposes, with OpenVPN gateway machines only one possible use out of many others. The base system is easily enough extended via packages which allows the base system to be very compact while still allowing to do almost any kind of modification to the resulting system in the form of packages.

OGB Modes & Runlevels

An OGB Linux system has to main modes of operation: Production and Debugging. Usually one would make a different media for these two purposes and enable Debugging by adding the `debug` package. The other difference is, that OGB uses Runlevel 2 for Production and 3 for Debugging.

OpenVPN Gateway Builder

System Requirements

OGB should run on any modern Linux 2.6 system (though it has been developed on SuSE SLES9). Specifically OGB makes use of INITRAMFS as root filesystem. No testing has gone to running OGB on Linux 2.4 systems and none will be done, unless somebody will contribute this.

OGB uses the software installed on the build system. This software is expected to be installed at the usual standard locations (e.g. binaries are searched for in `$PATH`). OGB should be mostly totally agnostic of the exact software versions in use, though it might happen that on very old or very new systems some software looks different than what was used in developing OGB (which is SuSE 9.3 Professional and SuSE Linux Enterprise Server 9 SP1)

OGB uses syslog-ng for logging which is more flexible and reliable than standard syslog. If your build system does not use syslog-ng, you must install syslog-ng (but not neccessarily use it) to be able to have logging in your OGB systems. There is also no error checking done in this regard.

The basic software requirements of OGB, which are needed to build a running system, should be met by virtually any functioning Linux system. Many tools will be copied along if they are present, but OGB will still function without them (unless you need their functionality). To see the exact list of all programs probed for, look into the `ogb.sh` file.

Be aware, however, that many programs **you** would like to use might need some others installed. For example, to setup TAP/TUN devices one needs tunctl that is packages in uml-utilities on some distributions. You have to make sure yourself, that all required programs are indeed present on the build system.

Obtaining OGB & Installation

Download the current OGB ogb-1.4.tar.gz archive and unpack it under `/opt`, it will create the `/opt/ogb` directory that contains all OGB files. No further configuration required.

Other (historic and development) OGB versions:
ogb-1.0.tar.gz
ogb-1.1.tar.gz
ogb-1.3.tar.gz
ogb-20050718.tar.gz
ogb-20050719.tar.gz

OGB runs and creates files and directories only in the `/opt/ogb` directory. To remove OGB, simply delete this directory. The OGB configurations are also backed up onto the OGB ISOs, so that you can always recreate any given OGB system from the ISO or CD.

OpenVPN Gateway Builder

Basic Usage

OGB is invoked with the `/opt/ogb/ogb.sh` script:

```
ogb.sh [Options]  [packages ...]
OpenVPN Gateway Builder Version 1.4 / 2007-08-16
Build 3d038cb4a9d7d02bbe776a5005aa418a
Copyright (C) 2005-2007 Schlomo Schapiro
OpenVPN Gateway Builder comes with ABSOLUTELY NO WARRANTY; for details
see the GNU General Public License at http://www.gnu.org/licenses/gpl.html
Available Options:
-V                      version information
-t                      Trace OGB scripts
-r a.b.c-xx-yy          kernel version to use (current: 2.6.16.21-0.8-default)
-i                      create OGB ISO image in default path (/opt/ogb/ISO)
-I /some/path/here      create OGB ISO image in in specified path

Available Packages:
crond                   cron daemon for recurring processes
debug                   Build DEBUG version of OGB with SSH remote access and more tools
dhcpcd                  DHCP Client Daemon support
dhcpd                   DHCP Server support
ntpd                    NTP time synchronisation
openvpn                 OpenVPN tunneling software
ppp                     PPP and PPPOE support
snmp                    SNMP agent (net-snmp based)
sshd                    SSH remote access (only via authorized_keys) in runlevel 3 only
vmware                  Support for running OGB as a VMware Virtual Machine
Default Packages: openvpn
```

OGB always needs a configuration to operate on. Configurations are subdirectories of `/opt/ogb/gateways` which contain the configuration files for a specific gateway system:

```
# ls -RF /opt/ogb/gateways/sample
/opt/ogb/gateways/sample:
./        authorized_keys  modules     openvpn/          ssh_host_dsa_key.pub
../       hosts            network.sh  resolv.conf       ssh_host_dsa_key
HOSTNAME  iptables.conf    ntp.conf    PACKAGES

/opt/ogb/gateways/sample/openvpn:
./  ../  dh1024.pem  sample.conf  server.crt  server.key  tmp-ca.crt
```

The files are mostly the standard configuration files of a Linux system, some are only snippets which are appended to default skeleton configuration file (see below the detailed description)

OGB supports the concept of packages which can be added to a build. The basic OGB will only build a really small Linux basis which can't do anything (except syslogging to another host). Support for specific applications like NTP, OpenVPN or SSH is added via packages. Packages can also require more or other configuration files, for example the package ntpd requires `ntp.conf`.

Usually, the process to create a new VPN gateway consists of these steps:

1. Create a new configuration, e.g. `cd /opt/ogb ; cp -a gateways/sample gateways/my-config`
2. Adapt the configuration files to your needs, e.g. `vi gateways/my-config/{HOSTNAME,resolv.conf,hosts,iptables.conf,modules,network.sh}`
3. Drop your OpenVPN configuration into `gateways/my-config/openvpn`, all `*.conf` files will be started automatically. Don't forget to remove the sample OpenVPN configuration.
4. Make a debug build with `ogb.sh my-config debug`
5. Write the resulting ISO image (`ISO/my-config.iso`) onto a CD/DVD and test
6. Repeat these steps and modify your configuration till satisfied
7. Make a final "production" build (without the debug package) and run your gateway machine with it

Configuration Files

Consult the sample configuration for examples of the configuration files. All files mentioned here must reside in the gateway configuration directory. If some files are missing, then the functionality they provide will be missing, too (for example without a resolv.conf there won't be DNS). OGB does not do a sanity check of your configuration, that is *your* job.

Package	File	Description
base	HOSTNAME	The fully qualified hostname, e.g. gw1.my.domain
	hosts	The usual `/etc/hosts` file. It should contain all relevant entries except for localhost.
	modules	List of modules and their options, if needed. The modules and their dependancies are included in the OGB system and loaded with the respective options in the specified order.
	iptables.conf	IPTABLES configuration as created by `iptables-save`. All iptables features of the build machine will be available on the gateway system.
	network.sh	Setup networking and everything else. While beeing called network.sh, this script is the only user-supplied script run during the boot of the OGB system. Here one has to setup all networking (like NICs, bridging, VLANs, routing, ...) and do any other required initialization.
	resolv.conf	The usual `/etc/resolv.conf` to configure DNS.
	PACKAGES	Always include these packages. Same as giving the packages on the command line of `ogb.sh`. If you put `debug` here, then you will always get a debug build ...
ssh	authorized_keys	The usual authorized_keys file for root. Put all allowed keys there for remote access. The OGB system will allow remote access **only** via SSH DSA keys, not with a password (which does not exist on the OGB system).

	ssh_host_dsa_key ssh_host_dsa_key.pub	The host SSH keys. If they are not supplied with the configuration, then they will be generated on the first build.
	sshd_config	Optionally, the usual `/etc/ssh/sshd_config` file, in case you don't like the standard sshd configuration supplied (which is Protocol 2 and Key authentication only).
ntpd	ntp.conf	Addon to `/etc/ntp.conf` which deals with your time sources. The default ntp.conf is very restrictive and doesn't allow anything.
openvpn	openvpn/*.conf	OpenVPN configuration files and their required cryptographic material and scripts. Be careful with using scripts, as the OGB system contains only a very limited choice of Unix utilities.
snmp	snmpd.conf	The usual `/etc/snmpd.conf` for the snmpd process. You really should supply this file and set your own syslocation, syscontatc and read only community and probably some access rules. The supplied snmpd.conf is just a default one with rocommunity public and no access restriction.
	mibs/*	Additional MIBS, if you need them. The snmp package includes a limited choice of MIBS which seemed most useful for a gateway without overdoing it. Everything needed for network and process monitoring is already included.

Some intended but untested things: ebtables/arptables binaries and modules are copied if found, but this has not been tested at all. 802.1q VLAN configuration with the vconfig command has not been tested in a production environment, though vconfig and the 8021q kernel module are copied.

OpenVPN Gateway Builder

Building

The build process used by OGB is very simple and straightforward:

- The build directory is cleared
- The skeleton directory is copied into the build directory and serves as a base
- A list of programs, libraries and configuration files to copy is assembled, the library dependancies are resolved and the files copied into the build directory
- The modules and their dependancies are copied into the build directory
- The isofs directory is cleared and filled with isolinux, the kernel and the CPIO archive of the build directory.
- The ISO image is created under ISO/

Packages basically extend this basic build process by adding a package skeleton, extending the lists of programs, libraries and files to copy and by calling 2 scripts, one before copying and one after. This is to provide hooks to both change the configuration files (used by the ssh module to generate the host keys) and to modifiy the OGB system in the build directory (used by most packages to add /etc/inittab entries for their function).

The most problematic part of OGB are probably the dependancy walkers that resolve the binary and module dependancies to include the needed libraries and modules. This code has been taken from SuSE's mkinitrd script and might have to be adapted to newer kernel versions or other changes. Till now it was always sufficient to copy over the respective code segments from the most current SuSE system. The code has also been verified to perform satisfactory on Debian and Red Hat systems.

Booting & Running an OGB system

ISOLINUX is used to boot the OGB system from CD. A stable version is included in the OGB distribution as there is no standard on installing SYSLINUX/ISOLINUX under Linux and almost each Linux distribution does it differently. In any case, the ISOLINUX binary is not run **in** the OGB system, but rather use to start it.

The OGB Linux runs off an INITRAMFS ramdisk which is initialized by the boot loader. Thus there is no need in OGB to deal with block devices of any kind like hard disks, CD-ROM drives etc. Consequently, the build process does not copy any block device drivers to the OGB system and the OGB system cannot access any data on hard disks.

After booting the kernel, the normal System V init process starts, but there are no `/etc/init` scripts that run but rather a simple boot script `/etc/boot` which has to setup all the system. This boot script also calls `/etc/network.sh` which should setup the network stuff.

After the inital boot, init starts (and monitors) any application listed in `/etc/inittab` which is also the recommended place for other additions. Unfortunately some programs alway fork into background, so that init cannot monitor them. Otherwise init is the most simply monitoring tool and sufficient for most purposes.

Debugging & Logging

The build process writes a log to `log/ogb-<config>.log` (and keeps the previous log, too). Upon experiencing problems of any kind, it is always worthwhile to check that log file for errors, though someimes there are spurious errors from some subsystems that are "just normal" (for example about modules without depancies). However, serious errors will abort the build process with an error message.

In debug mode (enabled with the `debug` package), a shell is started on tty2 (reachable with ALT-F2) which allows some simple in-system debugging. Additionally the package installs and starts an SSH server to allow remote debugging. The package also copies a lot more tools and utilities to the OGB system, as it is otherwise very hard do debug it (e.g. without `ls` and other standard commands). In production mode there is no interactive access to the OGB system at all, as a security precaution

OGB uses syslog-ng for logging because it is more flexible and reliable than standard syslogd. In tests we found out that the large amount of logging that can be created by OGB during startup is oftenly partially lost when using remote syslog via UDP. syslog-ng TCP logging reliably solved this problem and even should allow logging to the remote end of a VPN tunnel (because messages are buffered to a certain extent).

Locally on the OGB system no logs are written to the filesystems, tty12 shows the current log instead. To keep the logs permanently, one must use remote logging via one of the many syslog-ng logging facilities. It is recommended to have a central loghost for this purpose.

In most cases the log output generated is sufficient to pinpoint any sudden trouble, though to debug boot and network or driver problems one usually will have to use a debug build and debug it directly at the OGB console.

Finally, SNMP support can be included via the `snmp` package which includes the snmp daemon. It has been tested on our build systems and can be used to monitor many different parameters of a running OGB system. Furthermore, custom scripts can be included at specific OIDs, please refer to the standard NET-SNMP documentation for more information about SNMP monitoring.

OpenVPN Gateway Builder

Introduction Overview Installation Usage How it works

The modular concept

The modular concept of OGB with a very slim base system and extensions to that in the form of packages allows for a very flexible build solution for various needs. The packages are very simple and shouldn't be compared to RPM or DEB packages and their complex abilities, but rather to a very simple building plan for OGB extensions.

If you want to change how OGB behaves it is recommended to create a custom package instead of modifying the OGB script, so that your modification will be compatible with further updates of OGB. Of course some things can be done only in the base system (like fixing bugs for broken things like missing libraries), but in most cases a package is able to do the job and should then be used.

Please also send your packages back to the maintainer, so that they can be included in the next OGB distribution. packages should contain only configuration files, as all binaries and data file should be taken from the build system.

Making your own packages

The recommended way of adding new functionality to OGB is by creating a new package, which is very easy. Basically a package consist of a skeleton directory, lists of programs, libraries and files to copy and a prepare.sh script run before copying and a configure.sh script run after copying. Packages can also demand other packages which will then be included (for example, debug includes ssh).

Basically a package is the same as the main OGB system, with a skeleton directory of its own and lists of files to copy to the OGB system. The difference beeing that with packages these lists are maintained in text files and not hard coded into the OGB build script.

The `debug` and `snmp` packages can serve as a full example as they use most features available for a package. The order the packages are applied is the one given on the command line, included packages are appended to this list.

Other uses for OGB

OGB, besides beeing named after OpenVPN, could be useful also for other purposes where one needs to build a maintainable, small and secure mini Linux system. The OpenVPN package can be excluded from the default configuration and then OGB will just build a very basic Linux system. By adding other custom packages one can then create any Linux system desired.

OpenVPN Gateway Builder

Introduction | Overview | Installation | Usage | How it works

Troubleshooting

If the debugging and logging ideas do not solve your problem, then you might to find a solution by (please in this order)

- reading the FAQ below
- asking on the mailing list (*Sorry, the mailing list is down at the moment*)
- ask the maintainer at ogb @ schlomo . schapiro . org for help. I will answer emails as well and soon as I can. For urgent cases, there is also commercial support available, please mail to ogb-commercial @ schlomo . schapiro . org for inquiries.

F.A.Q.

OGB works, but my OpenVPN tunnels don't work !
Make sure that your OpenVPN tunnels work before using OGB. OGB does not help or hinder your tunnels anyhow differently than a "normal" Linux system, though you have to do all the network setup like IP adresses or routing yourself. There is - so far - no known case where OGB made OpenVPN not work, it was always a configuration error.

How can I create the SSL certificates to use with OGB and OpenVPN ?
I oftenly use the very nice TinyCA for this purpose. One could also use the eaysrsa package included with OpenVPN.

What about a GUI ?
A GUI is one of the TODOs for OGB, it makes most sense in a large scale environment where one has to manage lots of OpenVPN gateways. A Web-GUI would be also the best way to let people make their own personalised OGB system according to some standards.

If you are interested in this, please contact me to exchange ideas. I could also develop the GUI as a project for you.

What about CRL management ?
This is also on the TODO list. Basically one would make a package that would download CRLs regularly as OpenVPN rereads the CRL at each new connection. If you decide to write such a package, please submit it for inclusion. I can also provide some ideas how to do it.

What about OpenVPN plugins ?
Make a package for them as you need them. Most modules need other software (like radius or PAM) which is currently not part of OGB and most people do very site-specific stuff with modules.

How do I enable remote Logging ?
Create a `syslog-ng.conf` in the gateway directory that reads like this:

```
destination loghost { tcp("192.168.1.2" port(51415) ); };
log { source("src"); destination("loghost"); };
```

This would send all log data via TCP to my log host. Consult the syslog-ng manual on the build system for more information about the syslog-ng configuration.

How can I testdrive an OGB system ?
Put the CD into a computer and boot it up :-). One can of course also use a VMware virtual machine and now there is the free VMware Player which one can use for this purpose. Simply use this simple virtual machine. You might also include the `vmware` package and the `pcnet32` module in this case.