

Scratchbox Remote Shell

Timo Savola

`tsavola@movial.fi`

Scratchbox Remote Shell

by Timo Savola

Revision history

Version:	Author:	Description:
2005-02-08	Savola	Based on <i>Device tools</i>

Table of Contents

1. Introduction to sbrsh	1
1.1. Changes from version 1.4 to version 6.....	1
2. Using the daemon	2
2.1. Configuration	2
2.1.1. Basic options	2
2.1.2. Advanced options	2
2.1.3. Mount options.....	3
2.2. Authentication	3
3. Using the client	5
3.1. Configuration	5
3.2. Running	6
4. Debugging	7
5. Building sbrsh	8
6. Security issues	10
References	11
A. sbrsh 6.6 README	12

Chapter 1. Introduction to sbrsh

The Scratchbox Remote Shell is a remote command execution system similar to rsh and ssh. It is designed with slow devices and Scratchbox's [1] special requirements in mind. It supports common types of program execution (including terminal emulation), but it is optimised for non-interactive usage. The communication happens on a TCP/IP connection and is not encrypted—sbrsh is meant to be used only on trusted networks (such as a company's LAN or an USB network between a PC and a handheld device).

The server (sbrshd) is run on a device having the same CPU architecture as the compilation target that is being used in Scratchbox. It executes the commands issued by the client (sbrsh) inside a “sandbox” that is created by mounting network filesystems (typically exported by the host that runs the client) and binding local directories (such as /dev).

Sbrshd contains support functionality that makes network-transparent fakeroot [2] sessions possible. Due to this sbrshd requires fakeroot header files to compile.

Sbrsh is released under the GNU General Public License version 2 [3].

1.1. Changes from version 1.4 to version 6

- The version numbering was changed so that the major version corresponds to the protocol version and the minor version is the build number.
- The insecure password is no longer used. Instead, there is an option for using identd to authenticate connections. (See Chapter 6 for more information.)
- Sbrshd no longer needs normal user accounts for anything: the access rights are configured in /etc/sbrshd.conf and the sandbox directories are created at /var/sbrshd/. This also means that sbrshd does not need to validate login shells.
- The host-side umask setting automatically affects the target-side environment.
- Sbrshd can be run in “sandboxless” mode where it runs commands in the system environment instead of a sandbox.

Chapter 2. Using the daemon

Installing Scratchbox [4] has instructions for installing the sbrsh daemon. This chapter assumes that you have already installed sbrshd on a device. Using the init script is recommended but not required. This document will deal with the init script where applicable, but the init script's options are similar to sbrshd's command-line options.

2.1. Configuration

Sbrshd can be configured by editing the the variables at the top of `/etc/init.d/sbrshd`.

2.1.1. Basic options

PORT

The TCP port number for the daemon. Default is 1202.

IDENT

If true, an identd lookup is done to verify that the connections are initiated by authorised users. Default is false, but it is recommended that you enable this if it is practical in your working environment. See Chapter 6 for more information.

DEBUG

If true, a detailed debug log will be written into file `/tmp/sbrshd-PORT.log` where `PORT` is the value of the `PORT` option. Default is false. You can open and close the debug log also at runtime—see Chapter 4 for instructions.

EXPIRATION

The duration in minutes to keep mount points mounted in the sandbox directories. Default is 15 minutes. Value 0 causes unmounting immediately after command execution and value “none” means that unmounting is done only when sbrshd is stopped.

SBRSHD

If you have installed the sbrshd binary to a directory other than `/usr/sbin`, or you want to run several versions of sbrshd, you can specify the path to the executable here.

PIDFILE

The process ID of the daemon is stored in this file while it is running. If you are running several sbrshd's, you should use different files here.

2.1.2. Advanced options

LOCAL_ONLY

If true, the daemon listens to connections only from the localhost. Default is to listen to connections from everywhere.

This option is provided for possible future needs: if NFS is replaced with a secure alternative, it might also be sensible to tunnel sbrsh connections through ssh.

NO_SANDBOX

If true, the daemon does not create a sandbox directory and chroot to it before executing the command. Default is to use a sandbox.

Even when not using a sandbox, it is still possible to mount NFS filesystems and bind directories, but that will affect the system environment. If one would want to use the sandboxless mode, one would typically not define any mounts either.

ALLOW_ROOT

If true, the daemon allows the root user to execute commands. This is generally a bad idea, so it's best to leave this disabled. The option is provided in case someone really *really* needs to do this.

Note: You can use sbrsh within a fakeroot session even when this option is disabled. The commands will be run using normal privileges in a fakeroot session.

2.1.3. Mount options

Sbrshd uses the `/bin/mount` and `/bin/umount` commands for creating the sandbox. The mount command provided by the util-linux package uses the `--bind` option for binding directories, but this may not be the case with all versions of mount. For example Busybox [5] uses the `-obind` option instead. Sbrshd tries to auto-detect if the mount binary is Busybox, but it is also possible to specify the correct bind option. You can also specify which mount and umount binaries to use.

The init script does not directly support specifying these options. If you should ever need them, refer to sbrsh's README (Appendix A).

2.2. Authentication

Sbrshd permits command execution only for connections that originate from IP-addresses that are listed in `/etc/sbrshd.conf`. It also checks that the user executing the command is listed after the IP-address, but this check is not reliable unless you have enabled the `identd` support (see Chapter 6).

The `sbrshd.conf` file lists one address per line and enumerates the allowed user accounts after it:

```
<address> <user1> <user2> <...> <userN>
```

The tokens are separated by arbitrary whitespaces. You can use the `#` character to start a comment.

The user list can be replaced with the `*` wildcard character which indicates that all users from the address are granted access. The address can contain the wildcard at its end. The line `"* *"` grants access to everyone.

```
# This is an example sbrshd.conf

192.168.0.1  bill steve larry
192.168.0.*  alice bob
10.*        guest
172.192.*   *
```

There is a shortcut for adding new users and/or IP-addresses to `sbrshd.conf`. You can use the following command:

```
# /usr/sbin/sbrshd add <user>@<address>
```

where `<user>` is a username or the `*` wildcard and `<address>` is a full IP-address or a pattern (as defined above).

Chapter 3. Using the client

3.1. Configuration

The remote execution hosts are defined in a configuration file that associates symbolic names to the hosts. The target name and the configuration file can be specified when invoking sbrsh. The default is to use the first target defined in the `~/ .sbrsh` file (where `'~'` is the user's home directory).

Note: When Scratchbox uses sbrsh to automatically run foreign binaries on a remote host, it first checks if the config file `/targets/TARGET.sbrsh` (where "TARGET" is Scratchbox's active compilation target) exists and only then falls back to sbrsh's default config file.

The target configuration also lists the NFS filesystems and local directories that are used to create the sandbox at the target device.

```
# This is an example .sbrsh

# The following is a typical target configuration:
ARM 192.168.0.202:1203
  nfs 192.168.0.200:/scratchbox/users/tsavola/targets/ARM /      rw,nolock,noac
  nfs 192.168.0.200:/scratchbox/users/tsavola/home      /home rw,nolock,noac
  bind /tmp      /tmp
  bind /proc     /proc
  bind /dev      /dev
  bind /dev/pts  /dev/pts

# You can also omit the mounts:
Sandboxless armdevice.office.company.com
```

As you can see from the example, the filesystem description must be indented by whitespaces and you can use the `#` character to start a comment. "ARM" and "Sandboxless" are target names, and "192.168.0.202" and "armdevice.office.company.com" are the corresponding remote host addresses. "1203" after the IP-address is the listening port of the sbrsh daemon.

The whitespace-separated parameters of the "nfs" filesystem entries are (from left to right): the NFS filesystem (typically exported by the Scratchbox host), the mount point inside the sandbox and NFS options. The parameters of the "bind" entries are: the system directory on the device and the mount point inside the sandbox.

Note: The order of the filesystem entries is significant. The root filesystem must be the first, and so on.

3.2. Running

You can run commands with `sbrsh` like this:

```
> sbrsh ls -l           # run a command with one argument
> sbrsh -d /tmp pwd    # specify the working directory
> sbrsh -t ARM true    # run a command on the ARM target
> sbrsh                # run $SHELL
```

See the README (Appendix A) for more details.

One notable difference to `ssh` is that `sbrsh` doesn't use the shell to execute commands, and thus does not read any profile or resource files. It creates the target environment by copying variables from the client-side environment. It also transfers the `umask` setting to the target environment.

Sometimes you may not want to pass some architecture-specific variables for the target binaries or want to change them. `Sbrsh` can override or unset variables in the remote end by using variables prefixed with “`SBOX_ENV_`”. The following examples should illustrate the behaviour:

```
> export SBOX_ENV_FOO=bar
> sbrsh env | grep ^FOO
FOO=bar

> export SBOX_ENV_PATH=(UNSET)
> sbrsh env
sbrsh server: Can't execute command: env (No such file or directory)
```

`Sbrsh` doesn't copy the resource limit settings to the remote environment, but they can be tuned with environment variables of the form “`SBOX_RLIMIT_RESOURCE`”:

- Limit total CPU time of the process to 10 seconds:

```
> export SBOX_RLIMIT_CPU=10
```

- Always create core dumps:

```
> export SBOX_RLIMIT_CORE=unlimited
```

Chapter 4. Debugging

The daemon reports all error conditions to syslog. It also supports a debug log where it writes lot of information about its state and what happens during command execution.

Debug can be enabled permanently in the init script (see Section 2.1). The debug log can also be opened temporarily using this command:

```
# /etc/init.d/sbrshd enable-debug
```

Debug log can be closed at any time using this command:

```
# /etc/init.d/sbrshd disable-debug
```

The log is written into file `/tmp/sbrshd-PORT.log` where `PORT` is the TCP port number the `sbrshd` listens to. (Default is `/tmp/sbrshd-1202.log`.)

Here is an example session:

```
# /etc/init.d/sbrshd enable-debug
Opening sbrshd debug log: done.
# cat /tmp/sbrshd-1202.log
07-01-1970 07:55:46.374 3392 DAEMON Debugging enabled
07-01-1970 07:55:46.375 3392 DAEMON sbrshd version 6.6
07-01-1970 07:55:46.375 3392 DAEMON Port: 1202
07-01-1970 07:55:46.375 3392 DAEMON Local only: no
07-01-1970 07:55:46.375 3392 DAEMON Sandbox: yes
07-01-1970 07:55:46.375 3392 DAEMON Ident: no
07-01-1970 07:55:46.375 3392 DAEMON Allow root: no
07-01-1970 07:55:46.375 3392 DAEMON Mount expiration: 900 seconds
07-01-1970 07:55:46.375 3392 DAEMON Checking for expired mounts
07-01-1970 07:55:46.376 3392 DAEMON Waiting for connection
```

In this example, “3392” is the process ID of the `sbrshd` daemon and “DAEMON” is the type of the process. `Sbrshd` has four types of processes that write log entries: `DAEMON` accepts connections and handles mounts, `HANDLER` handles I/O between the client and the command process, `RELAY` relays faked root messages and `COMMAND` “bootstraps” and executes the command.

Chapter 5. Building sbrsh

This chapter contains instructions for building sbrsh from source code. You shouldn't normally need to do that, since the sbrsh program is included in Scratchbox and all toolchains ship sbrshd binaries for their target architectures. See *Installing Scratchbox* [4] and *Scratchbox toolchains* [6] for more information.

Sbrsh can be cross-compiled inside and outside Scratchbox. Sbrshd requires fakeroot's header files which are currently only available in Scratchbox's toolchain packages. The sbrsh source package is available in the `/scratchbox/packages` directory in the Scratchbox installation, but it can also be downloaded from the Scratchbox website [1].

The following examples build sbrsh inside Scratchbox. The client is compiled for use in Scratchbox:

1. Extract the sbrsh source package:

```
[sbox-TARGET: ~] > tar xfz /scratchbox/packages/sbrsh-6.6.tar.gz
```

2. Make sure you have selected the HOST target:

```
[sbox-TARGET: ~] > sb-conf select HOST
```

3. Go to the source directory and build the sbrsh executable:

```
[sbox-HOST: ~] > cd sbrsh-6.6  
[sbox-HOST: ~/sbrsh-6.6] > make sbrsh
```

4. If you want to use your newly built sbrsh as a CPU-transparency method, you can install it under `/host_usr` and modify an existing target configuration using the `sb-conf` command:

```
[sbox-HOST: ~/sbrsh-6.6] > mkdir -p /host_usr/bin  
[sbox-HOST: ~/sbrsh-6.6] > cp sbrsh /host_usr/bin/  
[sbox-HOST: ~/sbrsh-6.6] > sb-conf setup TARGET --cputransp /host_usr/bin/sbrsh --force
```

The server is cross-compiled for a preconfigured target:

1. Select the target your wish to compile sbrshd for:

```
[sbox-HOST: ~/sbrsh-6.6] > sb-conf select ARM
```

2. Go back to the source directory and clean it:

```
[sbox-ARM: ~] > cd sbrsh-6.6  
[sbox-ARM: ~/sbrsh-6.6] > make clean
```

3. Build the sbrshd executable using fakeroot's headers:

```
[sbox-ARM: ~/sbrsh-6.6] > make sbrshd \  
CPPFLAGS="-I/scratchbox/device_tools/fakeroot-1.2.3/arm-gcc-3.3.4-glibc-2.3.2/include"
```

4. If you are using the Debian devkit, you can also build a binary package for Debian:

```
[sbox-ARM: ~/sbrsh-6.6] > dpkg-buildpackage -rfakeroot -b
```

5. See *Installing Scratchbox* [4] for sbrshd installation instructions.

Chapter 6. Security issues

sbrsh is insecure for the following reasons:

1. the communication is not encrypted;
2. user authentication is not reliable unless `identd` is used;
3. sbrsh allows the mounting of arbitrary NFS filesystems.

Note: NFS in itself is also insecure.

Point 1. Sbrsh does not apply encryption on the TCP communication—after all, sbrsh is a development tool and designed to run on slow systems. You could use an ssh tunnel to encrypt the communication, but that would still leave the NFS communication unencrypted. (Also, the *fakeroot relay* implemented by sbrshd does not currently support this.)

Point 2. There is no guarantee that the username sent by the client is correct. If you (the administrator of a network containing Scratchbox workstations and sbrshd devices) would like to restrict access to sbrshd reliably (possibly due to Point 3), you should install an `identd` on the workstations and enable `identd` lookup in sbrshd (see Section 2.1.1). sbrsh has been tested with the “gidentd” Debian package [7].

Point 3. You can specify any NFS server and path in sbrsh’s target configuration. This is not a bug: If Scratchbox is installed on an NFS server, the Scratchbox itself runs in your workstation but the filesystems are exported by the server. In this case, your workstation connects to the sbrshd device and asks it to mount filesystems from the NFS server, which is correct. Unfortunately this also means that you can mount your coworker’s filesystems...

These points imply that you should use only dedicated devices for running sbrshd with no sensitive material (such as private keys) and you should make the device available only to your development team on a local network. But don’t be afraid; you should be safe when following these guidelines.

References

- [1] *Scratchbox website* (<http://scratchbox.org/>).
- [2] *Fakeroot in Scratchbox* (<http://scratchbox.org/documentation/docbook/fakeroot.html>), Timo Savola.
- [3] *GNU General Public License* (<http://www.gnu.org/copyleft/gpl.html>).
- [4] *Installing Scratchbox* (<http://scratchbox.org/documentation/docbook/installdoc.html>), Valtteri Rahkonen.
- [5] *Busybox* (<http://busybox.net/>).
- [6] *Scratchbox toolchains* (<http://scratchbox.org/documentation/docbook/toolchain.html>), Ricardo Kekki.
- [7] *Debian — gidentd* (<http://packages.debian.org/gidentd>).

Appendix A. sbrsh 6.6 README

General Information =====

Scratchbox Remote Shell is an rsh/ssh-like utility for Linux that supports terminal emulation, automated mounting of network shares, chroot, etc.

sbrsh is distributed under the General Public License version 2. See COPYING for the full license.

Client Usage =====

Execute a remote command:

```
sbrsh [-t|--target <target>]
      [-c|--config <path>]
      [-d|--directory <dir>]
      [<command> [<arguments>]]
```

Mount or unmount all filesystems of a target:

```
sbrsh [-t|--target <target>]
      [-c|--config <path>]
      --mount|--umount
```

--target	symbolic name of the target configuration; the first target in the config file will be used by default
--config	specifies the absolute path to the user's configuration file (default is "\$HOME/.sbrsh")
--directory	the user's current directory at the target device (default is "/")
command	name of the binary to be executed (may be looked up from PATH); if command is not specified, the SHELL environment variable is used; if SHELL is not set, "/bin/sh" is used
arguments	zero or more arguments for the command

Daemon Usage =====

```
sbrshd [-p|--port <port>]
       [-l|--local-only]
       [-n|--no-sandbox]
       [-i|--ident]
       [-r|--allow-root]
       [-d|--debug <path>]
       [-e|--mount-expiration <minutes>|none]
       [-m|--mount-bin <path>]
       [-u|--umount-bin <path>]
```

```
[-t|--mount-tab <path>]
[-b|--bind-opt <options>]
```

```
--port          sets a custom port number (default is 1202)
--local-only    makes sbrshd only accept connections from the local host
--no-sandbox    does not create a sandbox directory, create mounts or chroot
--ident        uses identd for validating incoming connections
--allow-root    allow commands to be run with user and group id 0
--debug        enables debugging to a log file
--mount-expiration specifies the number of minutes to wait before expiring
                unused mount points (default is 15); 0 means that filesystems
                are unmounted immediately after commands exit; "none" means
                that filesystems are unmounted only when sbrshd exits
--mount-bin     specifies the mount binary path (default is "/bin/mount")
--umount-bin    specifies the umount binary path (default is "/bin/umount")
--mount-tab     specifies the mount table path (default is "/etc/mtab")
--bind-opt     specifies the option used when binding a path to a mount point
                (default is "--bind", or "-obind" if mount binary is Busybox)
```

The pid of the daemon process is printed to stdout. The command exits immediately.

The debug log can be opened and closed at run-time by sending the daemon process the SIGUSR1 (open log) and SIGUSR2 (close log) signals. If the debug log was not opened at startup with the --debug option, the debug log will be written to "/tmp/sbrshd-<port>.log" (where <port> is the value of the --port option).

Client Config
=====

The client configuration file lists all known TARGETs (see Client Usage). The first line of a TARGET block must not contain whitespaces before the name of the TARGET. The subsequent lines must be indented. "#" is a line end comment character.

The layout of the first line:
 <target> <ip>[:<port>]

The subsequent lines define the mounts needed by the TARGET (TYPE is either "nfs" or "bind"):
 <type> <share/path> <point> [<nfs options>]

Here's an example configuration:

```
ARM 172.16.6.76:1202
nfs 1.2.3.4:/playground / rw,nolock,noac
bind /dev /dev
bind /proc /proc
bind /var/tmp /tmp
```

Daemon Config =====

The daemon configuration file is /etc/sbrshd.conf. It lists all client IP addresses and user accounts that are granted access. The user account list can be replaced with "*" to permit all users from a host. The "*" wildcard can also be used at the end of an IP address. "#" is a line end comment character.

The layout is:

```
<ip> <username1> <username2> <...> <usernameN>
```

Here's an example configuration:

```
172.16.6.10 john bob alice
172.16.6.85 *
172.16.6.* guest
192.168.* *
```

Environment Variables =====

The command execution environment at the target device can be controlled via a few environment variables.

"SBOX_ENV_" prefix will be stripped from all variables having one. If a corresponding variable without the prefix exists, it will be overridden. If the variable's value is "(UNSET)", the corresponding variable will be removed from environment. For example the dynamic linker can be controlled this way (via the LD_* variables) without affecting the sbrsh client itself.

The resource limits can be set using variables with the "SBOX_RLIMIT_" prefix. The value can be either an integer or "unlimited". The supported setting are:

SBOX_RLIMIT_CPU	CPU time in seconds
SBOX_RLIMIT_FSIZE	max filesize
SBOX_RLIMIT_DATA	max data size
SBOX_RLIMIT_STACK	max stack size
SBOX_RLIMIT_CORE	max core file size
SBOX_RLIMIT_RSS	max resident set size
SBOX_RLIMIT_NPROC	max number of processes
SBOX_RLIMIT_NOFILE	max number of open files
SBOX_RLIMIT_MEMLOCK	max locked-in-memory address space
SBOX_RLIMIT_AS	address space (virtual memory) limit