

NAME

`sc_remoted` — interact with a collection of remotely controlled scamper instances

SYNOPSIS

```
sc_remoted [-?46Dv] [-O options] [-M mux-socket] [-P [ip:]port] [-U unix-dir]
  [-R chroot-dir] [-C tls-ca] [-c tls-certificate]
  [-p tls-privatekey] [-m meta-file] [-e pid-file] [-Z zombie-time]
```

DESCRIPTION

The `sc_remoted` utility provides the ability to connect to a `scamper(1)` instance running remotely and interact with it by issuing commands and receiving results in warts format. The options are as follows:

- ? prints a list of command line options and a synopsis of each.
- v prints the version of `sc_remoted` and exits.
- D causes `sc_remoted` to operate as a daemon.
- 4 causes `sc_remoted` to only listen for IPv4-based connections.
- 6 causes `sc_remoted` to only listen for IPv6-based connections.
- O *options*
 - allows the behavior of `sc_remoted` to be further tailored. The current choices for this option are:
 - **allowgroup**: allow members of the unix domain socket's group to access to the unix domain sockets created by `sc_remoted`.
 - **allowother**: allow anyone on the system access to the unix domain sockets created by `sc_remoted`.
 - **debug**: print debugging messages to stderr.
 - **select**: use `select(2)` with all sockets, rather than `epoll(2)` or `kqueue(2)`.
 - **skipnameverification**: do not verify the monitor name, if presented, against the name in the certificate that the client presents if doing client TLS authentication.
 - **notimes**: do not preface debug output with a timestamp.
- P [*ip:*]*port*
 - specifies the IP address and port on the local host where `sc_remoted` should listen for incoming connections. If an IP address is not specified, `sc_remoted` will listen on all available IP addresses for incoming connections.
- M *mux-socket*
 - specifies location in the file system on the local host where `sc_remoted` should provide a multiplexed interface for access to remote hosts.
- U *unix-dir*
 - specifies the directory on the local host where `sc_remoted` should place individual unix domain sockets corresponding to individual remote hosts.
- R *chroot-dir*
 - specifies the directory on the local host where `sc_remoted` should `chroot(2)` to on start. If used, then other arguments that specify locations are relative to this directory. See the security considerations section for further documentation.
- C *tls-ca*
 - specifies the certificate authority certificate file in PEM format for `sc_remoted` to use to verify client certificates.
- c *tls-certificate*
 - specifies the server certificate file in PEM format to advertise to remote `scamper(1)` instances.

- p *tls-privatekey*
specifies the private key file in PEM format that corresponds to the certificate file. This key should have a passphrase. **sc_remoted** will prompt for the passphrase when starting up.
- e *pid-file*
specifies the name of a file to write the process ID to.
- m *meta-file*
specifies the name of a file containing meta data for remote hosts.
- Z *zombie-time*
specifies the length of time **sc_remoted** will retain state for a disconnected *scamper(1)* instance, allowing it to resume. By default **sc_remoted** retains state for 15 minutes. **sc_remoted** can wait for up to three hours for a remote scamper instance to resume.

EXAMPLES

The intended use of the remote control socket built into *scamper(1)* is as follows. A central server with IP addresses 192.0.2.1 and 2001:db8::1 runs a **sc_remoted** process listening on a port for remote scamper process, placing control sockets in a specified directory:

```
sc_remoted -P 31337 -U remote-socket-dir
```

Then, a remote host with IP address 198.51.100.55 runs scamper and connects to the remote controller:

```
scamper -R 192.0.2.1:31337
```

The **sc_remoted** process places a unix domain socket in the directory corresponding to the remote process. The name corresponds to the source IP address and port the remote scamper process connected to controller with. If the scamper process used source port 1025, then the unix domain socket's name will be

```
remote-socket-dir/198.51.100.55:1025
```

If a second remote host with IP address 2001:db8:1234::1 runs scamper and connects to the remote controller:

```
scamper -R [2001:db8::1]:31337
```

The same **sc_remoted** process will place another unix domain socket in the directory corresponding to the remote process. If the scamper process used source port 1026, then the unix domain socket's name will be

```
remote-socket-dir/2001:db8:1234::1.1026
```

If scamper is started with **-M** monitor-name, then it will pass the monitor name `sc_remoted`, which will use it in the unix domain socket's name. For example, if scamper is started as follows:

```
scamper -R [2001:db8::1]:31337 -M foo.bar
```

then the unix domain socket's name will be

```
remote-socket-dir/foo.bar-2001:db8:1234::1.1026
```

Because providing a unix domain socket per remote scamper process scales poorly in its use of file descriptors, it is recommended that **sc_remoted** provides a multiplexed interface to the remote scamper instances over a single unix domain socket, as follows:

```
sc_remoted -P 31337 -M mux-socket
```

It is possible to simultaneously provide a mux-socket and individual unix domain sockets in a separate directory, as follows:

```
sc_remoted -P 31337 -M mux-socket -U remote-socket-dir
```

PROVIDING METADATA TO REMOTE CONTROLLER USERS

sc_remoted can provide metadata for remote scamper instances to users of the multiplexed interface. *libscamperctrl(3)* provides interfaces to use the multiplexed interface and obtain metadata at runtime. To associate metadata with remote scamper instances, the instances must be started with unique monitor-name

values, such as

```
scamper -R 192.0.2.1:31337 -M foo.bar
```

which self-identifies as foo.bar. Given a **sc_remoted** process started as follows:

```
sc_remoted -M mux-socket -m meta.txt -P 31337
```

with meta.txt containing metadata formatted as follows:

```
foo.bar asn4 64504
foo.bar asn6 64504
foo.bar cc nz
foo.bar st wko
foo.bar place Hamilton
foo.bar latlong -37.7875184,175.2783528
foo.bar shortname foo
foo.bar iata hlz
foo.bar tag os:freebsd
foo.bar tag hardware:pi4
```

then users can programmatically identify that the remote system named foo.bar is located in New Zealand, and is a Raspberry Pi4 running FreeBSD.

USING TRANSPORT LAYER SECURITY

sc_remoted and scamper support the use of transport layer security (TLS) using OpenSSL to authenticate and encrypt communications between **sc_remoted** and scamper. To use this support requires a certificate signed by a certificate authority. Scamper will verify the certificate presented by **sc_remoted** and disconnect if the certificate presented by **sc_remoted** cannot be validated.

Generating a certificate that will be accepted by scamper requires you to create a certificate request and pass it for signing to a certificate authority. To generate a private key in file remotpriv.pem, and a request to sign the key in remotereq.pem:

```
openssl req -new -keyout remotpriv.pem -out remotereq.pem
```

and then send the remotereq.pem file to the certificate authority for signing. Do not send remotpriv.pem; that key must remain private to you. When openssl prompts for a passphrase, choose a passphrase that is unique and keep the passphrase secret. When your chosen certificate authority signs your private key, it will return a file which we will call remotecert.pem. Both remotecert.pem and remotpriv.pem are required parameters to **sc_remoted** to enable TLS support:

```
sc_remoted -P 31337 -U remote-socket-dir -c remotecert.pem -p
remotpriv.pem
```

and then run scamper as follows:

```
scamper -R example.com:31337
```

sc_remoted can also require that scamper present a certificate during the TLS handshake with the **-C** parameter:

```
sc_remoted -P 31337 -U remote-socket-dir -c remotecert.pem -p
remotpriv.pem -C remoteca.pem
```

In this case, **sc_remoted** requires that the scamper instance passes valid certificate signed by remoteca.pem, and that the certificate contains a monitor-name matching the monitor-name subsequently provided by scamper to **sc_remoted**. The scamper-side of this process looks like:

```
scamper -R example.com:31337 -O client-certfile=cert.pem -O
client-privfile=key.pem -M foo.bar
```

SIGNAL HANDLERS

sc_remoted installs handlers for three signals: SIGINT, SIGTERM, and SIGHUP. SIGINT and SIGTERM cause **sc_remoted** to exit gracefully. SIGHUP causes **sc_remoted** to reload the TLS certificate and private key, without interrupting existing TLS connections, and reload the metadata file.

SECURITY CONSIDERATIONS

sc_remoted should always be run by a non-root user.

sc_remoted provides an optional *chroot(2)* capability, which is available when *unveil(2)* is unavailable, and it is desirable that **sc_remoted** has restricted access to files available on the system. If **sc_remoted** is run on OpenBSD, which provides *unveil(2)*, you do not need to read further.

The *chroot* option requires that the process is started with capabilities. On Linux **sc_remoted** should be provided the CAP_SYS_CHROOT capability. On other systems, **sc_remoted** should be installed setuid root. Some systems have documented methods to escape a *chroot* where directories beneath a *chroot* are moved out of the *chroot* while the process is running. For this reason, use of the *unix-dir* option to store individual unix domain sockets for remote scamper instances in a directory beneath the *chroot* is discouraged.

SEE ALSO

libscamperctrl(3), *scamper(1)*, *sc_attach(1)*, *sc_wartsdump(1)*, *warts(5)*, *openssl(1)*, *chroot(2)*, *pledge(2)*.

AUTHORS

sc_remoted was written by Matthew Luckie <mjl@luckie.org.nz>.