Scriptroute Administrator's Guide

Neil Spring

March 12, 2005

Contents

Chapter 1

Introduction

Scriptroute is a distributed platform for flexible, lightweight network measurements. Scriptroute servers host measurement scripts that send network probes, like TTL-limited UDP messages or ICMP timestamp requests, and collect responses, like ICMP time exceeded or timestamp reply. Hosted measurement scripts can then compute performance attributes or send additional probes to further uncover network properties.

This manual is a guide to the configuration of the Scriptroute server. It also describes Scriptroute from an administrator's perspective, including both security aspects and what processes and files are used. The Scriptroute Scriptwriter's Guide describes writing measurement scripts and using Scriptroute for distributed network measurement.[1]

1.1 Architecture

Scriptroute can be broken into three sets of machines: clients, servers, and home. The home servers[2] maintain a list of other servers and destination-specific policy. Clients connect to a home server using DNS or HTTP to discover a Scriptroute server within a geographic region or autonomous system (a certain provider's network). Clients can then submit measurement scripts using HTTP Post to a front-end webserver running on a server. The front-end passes the measurement to a safe-mode ruby interpreter that runs in a resource-limited chroot sandbox. The interpreter interacts with the Scriptroute daemon, which decides which and when packets should be sent through a raw socket.

The Scriptroute daemon and interpreter software can also be used locally. That is, a user on the local machine can run an interpreter exempt from some of the resource and safe-mode limitations imposed by the unauthenticated execution model of script submission by HTTP. This can be handy for debugging the installation and for developing new scripts, and also provides a different model for network measurement tools that might be installed on your machine, reducing the number of tools that need to setuid root.

1.2 Ruby

Scriptroute measurement scripts are based on Ruby, a general purpose, object oriented scripting language. We chose Ruby because it is easily embedded, easily sandboxed, and easy to use. For a brief description of Ruby, please see http://www.ruby-lang.org/en/whats.html. "Programming Ruby" by Thomas and Hunt is a more detailed guide and reference, and is available online [**?**]. If you just want to modify an existing measurement, learning Ruby should not be a hassle. If you instead need to write a measurement from scratch, you'll probably need to read the first few chapters of Programming Ruby.

The Scriptroute server runs as a small webserver that invokes a resource limited interpreter using CGI. Submitting a script is as easy as posting to a Web form, and tools are provided for remotely executing a script with arguments.

[1] At least, that's the goal. If there's stuff that you think should be described here, but isn't, please let us know. Contact information is at the end of this manual.

[2] Support for multiple "home" servers was added in late 2004.

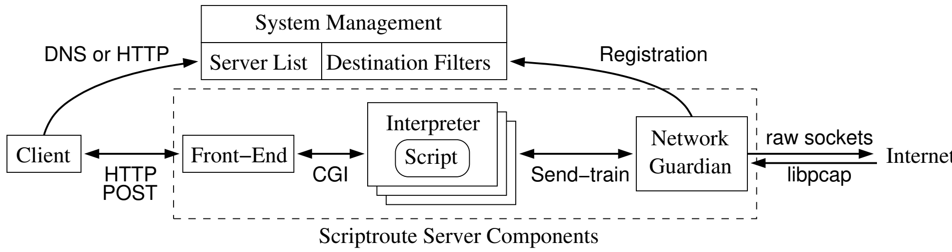

Figure 1.1: System architecture. I usually refer to the "Network Guardian" as "the Scriptroute daemon"; it has the name "scriptrouted." DNS policy queries (from an older vesion of this document) have been replaced with a custom registration protocol that disseminates the entire blacklist rather than query DNS for each destination.

The use of a scripting language that has not completely matured has caused some small pains within scriptroute—the interpreter may be a version 1.6 or 1.8 interpreter, with mildly different syntax and libraries. The newer version, 1.8, has fewer bugs, but 1.6 works fine. Further changes may happen in the future.

1.3 Security

Scriptroute is designed to be an open system: users of the system do not need authentication credentials, and Scriptroute hosts can be run by anyone. This influences the design of the system significantly, and limits the set of network measurements that can be supported.

Despite this design, some are more comfortable hosting servers when there is an audit trail back to at least some email address. Scriptroute cookies are given to users with valid email addresses so that a request can be bound to that email address. Send mail to cookies@scriptroute.org to get access to the list that maps cookie identifier to an email address.

1.3.1 Protecting the host

We support measurement scripts using a resource-limited, safe-mode interpreter running as nobody inside a chroot sandbox.

The interpreter's safe mode prevents access to file-system related system calls. This means that the interpreted script is not allowed to open files within the sandbox.

The kernel resource limits prevent the script from, for example, forking multiple threads, consuming significant memory, dumping a core file into the chroot, or using more than a few seconds of CPU time. The Scriptroute daemon enforces a limit on the number of interpreters that run concurrently, and the front-end Web server enforces a limit on the interpreter's runtime.

Running as nobody inside the sparse chroot we use appears to make the chroot unbreakable. A chroot environment can be broken if a process within the chroot can gain root and has access to certain files [**?**]. In our environment, to break the chroot, a script would have to:

1. Break out of the interpreter's safe mode that prevents files from being opened.

2. Find a way to create a directory as nobody, when the chroot is writeable only by root.

3. Find a way to call chroot when there's nothing in the chroot to escalate privilege to root. (the only binary in the chroot is the interpreter and the only additional files are logs).

One may simply not run the thttpd process, not install the ruby CGI interpreter, or configure the Scriptroute daemon to not register itself with the home server. All of these would provide further certainty that nothing bad will happen, but does not contribute to the global system.

1.3.2 Protecting the Network

Measurement traffic necessarily implies generating traffic that is abnormal: neither Web nor mail. The issue faced by Scriptroute is how to permit a wide variety of measurement probes, including timestamp requests, traceroute-like UDP packets and independent TCP acknowledgements, without supporting mischief. We apply three approaches to restricting traffic.

First, filters (in a "reverse-firewall") prevent sourcing potentially dangerous traffic. That is, we avoid sending fragments (that might be used for reassembly state exhaustion attacks or as a ping-of-death) and broadcast packets (that might be used as part of a smurf attack). We conservatively forbid traffic to low-numbered ports in the default policy, though in an ideal world, we wouldn't have to take such measures that are only intended to reduce complaints from system administrators who send mail whenever they are port-scanned. Destination-specific filters can be installed if a destination host complains about measurement traffic. Such a filter can be specific, e.g., "tcp port 80" which would prevent accesses to the remote Web server. It can also prohibit all traffic, e.g., "ip" traffic.

Second, rate limits are used to restrict the amount of traffic sent to any destination prefix, to restrict the amount of traffic consumed by Scriptroute overall, and to restrict the rate of SYN frames sent by an experiment. These rate limits protect different things. The destination-prefix rate limit is intended to prevent distributed denial of service attacks by reducing the degree to which a client can amplify their traffic. The overall rate limit provides a way for the server administrator to express that only a certain amount of traffic should be sent by Scriptroute. Public traceroute servers generally lack such a feature, making them easy to abuse. Finally, the SYN limit is intended to discourage the use of Scriptroute for port scanning or SYN flooding. Combined with the experiment duration limits above, sustaining a distributed flooding attack against any destination seems cumbersome and unlikely to be a worthwhile use of "script kiddie" time.

Third, in the event that a client abuses Scriptroute to generate some traffic the destination does not care for, logs are made of which client requested which packet. So, while clients are not authenticated, that does not imply that they are also anonymous. That is, traffic cannot be laundered through Scriptroute.

These three basic strategies protect the network, but there are possible nefarious uses for Scriptroute.

- A new attack, for which filters have not been developed, or that uses a sequence of packets that cannot be filtered statelessly.
- Port scanning through Scriptroute, aimed at high-numbered ports.
- Distributed flooding attacks that circumvent the destination rate limits through clever understanding of the network topology and routing.

If you are concerned about these potential dangers, do not host Scriptroute.

Chapter 2

Policy

There has been some confusion about what behaviors are permitted and what behaviors are forbidden, and how to circumvent any policy restrictions. “Policy” refers to any administrative limit that prevents a measurement script from doing anything that is otherwise permitted by the implementation. For example, packets are prohibited from being sent to the local subnet in the default policy, but the implementation doesn’t care what the destination is.

It is confusing because there are two convolved issues. First, why is behavior X prohibited? (“but I want to send TCP SYN packets to everyone in the internet!”) Second, which scriptroute component applies the policy?

The decomposition is straightforward. The scriptroute core daemon protects the network. It does not care who you are, whether you’re running locally (using the local interpreter) or remotely (via the web interface). This is a feature of Scriptroute’s security model. If it’s okay to do, anyone should be able to do it. If it’s not okay, it should be forbidden.

The interpreter protects the host. It is the environment for remote code execution and must apply resource limits and safe mode (no filesystem access) restrictions. Now, if you’re running locally, under your own user account, these restrictions are not necessary. You can disable them by using the “-u” (unsafe mode) and “-d” (no resource limits)[1] options. You can disable these options because it’s only your account that is hurt when a script does something unsafe. Using features enabled by the “-ud” option set will make it impossible to run your script remotely, which is why these features are disabled by default.

The front end web-server (thttpd) protects the availability of remote service. It will kill off any scripts executed on behalf of remote users after they’ve taken too long. This is a feature compiled into the webserver and is difficult to change or disable.

But I’m running as root, shouldn’t I be able to send whatever I want? To do so, you’ll need to change the policy in scriptrouted.conf to include what you want to do as “safe” behavior. You accept responsibility for any damage this change in policy causes. That means, if you violate your service agreement with your provider by sending bad traffic, don’t blame us.

The scriptroute daemon doesn’t automatically give you license to send whatever you want when running as root philosophically because unsafe traffic is unsafe traffic. The implementation doesn’t support any user-based authentication, so whether you’re running as an ordinary user, as root, or as nobody, makes no difference to the daemon, and consequently to the traffic you are permitted to send.

[1] The “-d” flag is needed when running under the debugger, hence its name.

Chapter 3

Installation

Scriptroute can be installed in two main ways. The preferred way is as part of a global network measurement infrastructure, proving service to unauthenticated clients in the network.

The paranoid among you may find the idea of running any code on your machine on behalf of another user frightening, so may want to install only the Scriptroute daemon so that only users with shell access are allowed to conduct network measurements.

3.1 Installed Files

Files may be installed in different places depending on how you installed Scriptroute.

3.1.1 /usr/bin: User executables

```
/usr/bin/sr-pingp
/usr/bin/sr-ally
/usr/bin/sr-ping
/usr/bin/sr-traceroute
/usr/bin/sr-sprobe
/usr/bin/sr-tcptraceroute
/usr/bin/srinterpreter
```

The sr- files are measurement scripts that use the local Scriptroute interpreter and daemon to send traffic. The local srinterpreter acts as `/usr/bin/perl` would for perl scripts. It has a man page, srinterpreter(1).

3.1.2 /usr/sbin: Daemons

```
/usr/sbin/dnskeeper
/usr/sbin/scriptrouted
```

The Scriptroute daemon is kept in /usr/sbin. It has a man page, scriptrouted(8). When running, the Scriptroute daemon listens only on localhost, which means that only local processes can request that packets be sent. The dnskeeper is used when running your own Home server using tinydns.

3.1.3 /usr/share/man: Basic documentation

```
/usr/share/man/man1/srinterpreter.1.gz
/usr/share/man/man8/scriptrouted.8.gz
```

Man pages for the interpreter and daemon. Additional pages will eventually be added for the different scripts.

3.1.4 /usr/lib/scriptroute: Supporting executables

```
/usr/lib/scriptroute/thttpd
/usr/lib/scriptroute/default_config
```

This copy of thttpd is one compiled for Scriptroute. A different version or copy of thttpd can be run instead; this version is bundled in case you have another Web server installed. It seemed cumbersome to distribute a package that depends on a Web server daemon that might conflict with one already installed on your system.

The default_config programs generates a somewhat sane configuration of Scriptroute, automatically generating some filters based on the local IP address and the local NTP system. The default configuration is described in Chapter 4.

3.1.5 /etc/scriptroute: Configuration files

```
/etc/scriptroute/scriptroute-thttpd.conf
/etc/scriptroute/scriptrouted.conf
/etc/scriptroute/dnskeeper.conf
```

Configuration files are kept in `/etc/scriptroute`. `scriptrouted.conf` is described in Chapter 4.

The thttpd configuration file, `scriptroute-thttpd.conf` is described fully at [?]. For our purposes, there are five relevant lines.

chroot The thttpd server will start itself up in the normal environment, then chroot to a directory described below. This style of scheme makes it relatively easy to construct a simple chroot.

port=3355 We chose to provide service on port 3355. Assertions that this is dumb welcome.

dir=/var/lib/scriptroute/web The new root directory. This is the root directory of the Scriptroute Web service (that is, what you would see if you looked at http://localhost:3355/)

logfile=/var/lib/scriptroute/web/log Where to log accesses. The logfile should be within thttpd's chroot.

cgipat=/cgi-bin/* How thttpd recognizes cgi scripts. In this case, we're saying that those files in cgi-bin are to be executed and not just served up.

The dnskeeper configuration file is unlikely to be relevant for you; it configures the dnskeeper daemon at the heart of the "Home" server in the architecture.

3.1.6 /etc/logcheck: Log filters

In the Debian package, logcheck entries are added:

```
/etc/logcheck/ignore.d.workstation/scriptroute
/etc/logcheck/ignore.d.server/scriptroute
```

Logcheck is a system for watching the logfiles of a system and sending mail with exceptional events. These files list expressions of "normal" log file traffic that can be ignored.

3.1.7 /etc/init.d: Daemon startup

```
/etc/init.d/scriptroute
/etc/init.d/scriptroute-thttpd
```

You'll need a startup script like these. I note the files as they are placed for Debian. The thttpd front end server can be independently disabled by disabling the init.d file.

3.1.8 /var/log/scriptroute: Sent packets log

`/var/log/scriptroute/packets`

This is a logfile consisting of packets sent by Scriptroute. In includes the client that requested packets be sent, which prevents anonymously bounce packets through Scriptroute. One can trace after the fact where a packet was sent from.

The packets log is owned by the user `scriptrt`, which isn't owned by nobody only to prevent anything inside the thttpd chroot from being writable by nobody.

This log file should be rotated as others on your system. Sending `scriptrouted` a HUP signal will cause it to reopen the log after rotation, much like other daemons.

3.1.9 /var/lib/scriptroute/web: Front-end chroot

```
/var/lib/scriptroute/web/cgi-bin/srrubycgi
/var/lib/scriptroute/web/var/log/scriptroute
/var/lib/scriptroute/web/index.html
```

srrubycgi is a statically linked Scriptroute interpreter with resource limits enabled. Static linking is generally discouraged, since it means that security fixes in libraries imply that dependent packages must be recompiled. In this case, we statically link the interpreter to simplify the construction of the chroot.

The `/var/log/scriptroute` inside that directory is a directory for a bind mount to the actual `/var/log/scriptroute`. This works on Linux; something interesting will have to be discovered to continue using the chroot on non-Linux systems.

Finally, the index.html file is just a reference to the main Scriptroute Web.

3.2 Dependencies

Before installing Scriptroute, you will probably want to install the ruby interpreter and libpcap. If you install the ruby interpreter from a distribution, also download the -dev package that contains static libraries and header files.

If these dependencies are not fulfilled, the configure script will download and compile them on your behalf. Of course, then you're left with a machine that has no Ruby interpreter, which is just sad.

3.3 Source

Installation from source is not yet well supported. It usually builds fine, but you'll need to write your own init script pointing to the right place to get the daemons up and runnning.[1]

Fetch the latest distribution from http://www.scriptroute.org/source/. You should probably be used to this drill.

1. tar xvfz scriptroute-*.tar.gz
2. cd scriptroute-*/
3. ./configure
4. make
5. sudo make install

Where sudo should be replaced with whatever scheme you use to gain root (perhaps "su -c" instead).

On BSD-like systems, you will probably need to run "gmake" or "gnumake" instead of "make" above. The Makefiles have a handful of indispensible GNU-make-isms.

You will now need to copy or edit debian/init.d or redhat/init.d to find your binaries in /usr/local, or wherever they've been installed.

[1] Contributed init scripts for the default source install would be welcome

3.3.1 Options to ./configure

--disable-crypto Tell configure not to even look for libgcrypt. Use this option if you have libgcrypt installed but do not need a server built that supports Scriptroute cookies. This option may be useful if the version of libgcrypt you have installed is incompatible with (newer than[2]) the one Scriptroute expects.

--disable-static Tell configure not to try to build a statically-linked srrubycgi program. Use this option if your system lacks the .a files needed (e.g., Solaris) or if static linking just isn't working.

--enable-werror A development option to stop the build if a warning is found in some of the source directories. You probably don't want this flag on, but I use it all the time.

--with-dmalloc A development option to configure with dmalloc (memory debugging library) support.

3.4 Debian

Add the following to your /etc/apt/sources.list:

```
deb http://www.scriptroute.org/debian ./
deb-src http://www.scriptroute.org/debian ./
```

Then run:

1. apt-get update
2. apt-get install scriptroute

This package is developed using Sarge, built from a current pbuilder archive. If dependencies are incorrect for installing the provided binary package, you can first try:

1. apt-get update
2. apt-get -t unstable install scriptroute

Or you can build from source:

1. sudo apt-get build-dep scriptroute
2. apt-get –build source scriptroute
3. sudo dpkg -i ./scriptroute_*.deb

Where sudo should be replaced with whatever scheme you use to gain root (perhaps "su -c" instead).

And please be sure to check for somewhat regular updates.

A `scriptrouted.conf` is dynamically generated as the package is installed (in the postinst, if you're familiar with packaging internals). It should be edited.

3.5 RedHat

RPMs are at http://www.scriptroute.org/redhat/. As a redhat user, I assume you're clever enough to use this.

As an alternative, you may choose instead to build from source, then 'make rpm'. Or, you may choose to run `rpm -tb --clean scriptroute-*.tar.gz`, or `rpmbuild -tb scriptroute-*.tar.gz` which will automatically build a redhat package from a tarball. The command you use depends on the version of redhat you're running.

A `scriptrouted.conf` is dynamically generated as the package is installed (in the postinst, if you're familiar with packaging internals). It should be edited.

These packages are lightly tested. I am a Debian bigot.

[2] The configure script will ignore old versions automatically.

Chapter 4

Configuration

The default configuration of the system will be placed either in `/etc/scriptroute/scriptroute.conf` if installed as a package, or `/usr/local/etc/scriptroute/scriptroute.conf` otherwise. The default configuration values can be extracted by running the `default_config` program, which may be placed in `/usr/lib/scriptroute`. The default configuration file will include these configuration variables annotated with these descriptions.[1]

4.1 Annotated Default Configuration

We now go line by line through the default configuration file, and describe each configuration option in turn. Two entries are set by `default_config`, so sanity check at least: `Filter.listen_filter` and `Registration.ntp`. The rest are commented defaults.

4.1.1 Registration

The Registration parameters control how this scriptroute server announces itself as part of the global scriptroute system. The announcement is added into a directory that allows clients to find appropriate servers quickly. You‘ll want to change most of these parameters, which is why they‘re listed first.

Registration.continent na

Two-letter continent code (na, eu, sa, af, as, au, at).

Registration.country us

Two-letter country code (us, uk, it ...).

Registration.admin nspring@scriptroute.cs.umd.edu

Your email address should we need to get in touch with you urgently, either to disable the system, or in the event that you‘re running an obsolete version.

Registration.ntp NTP-stratum4

Whether the machine runs ntp (eg. NTP-stratum2). The default configuration is likely to have the result of running ntptrace on your machine. If you don‘t have ntptrace, but do have ntp running, well, find some place that does have ntptrace and run it with your machine name as the first argument. If you‘re not running NTP, say “none” here. NTP can help some measurements and hurt others.

[1] They are reproduced here in an easy to read form because I don’t believe in configuration files as documentation: one should be able to preserve a configuration file from a previous version of a tool and get sane behavior—coupling documentation with the configuration file means that an upgrading administrator has to choose between manually merging a configuration and having complete documentation.

4.1.2 Operation

Operation parameters affect how Scriptroute interacts with the local operating system and define some magic parameters. Operation is an umbrella class that houses some diverse parameters that don't have their own class.

Operation. syslog_facility daemon
: Which syslog facility (e.g., "daemon", "local3") to use for debugging and error logs. Scriptroute uses syslog for message logging, not for packet logging. In the event that you can run scriptroute, but can't read back from syslog (say, you're running on planetlab or in some other way as an ordinary user), this can be set to a full path, that is, one that starts with '/'.

Operation.use_cap_net_raw_syscall False
: (Deprecated/Obsolete) Whether to use the planetlab specific temporary syscall to gain cap_net_raw. This will be removed at some point.

Operation.cache_dst_policy_minutes 5
: (Deprecated/Obsolete) How long to cache destination policies for. Destination policies are those filters put in place to appease destinations who forbid measurement traffic. The cache means that the destination filter doesn't need to be looked up on every sent probe packet

Operation.whois_server whois.ra.net
: (Deprecated/Obsolete) What whois server to use to figure out the origin AS for an IP address. This is used by the dns directory server (dnskeeper), not by the scriptroute system itself.

Operation.default_timeout_ms 3000
: What the timeout should be when a probe doesn't receive a response. Raising the timeout makes it more difficult to write tools that send packets to unresponsive or overloaded destinations. Lowering the timeout implies you know more about the network than does TCP.

Operation.default_timeout_randomization_range_ms 1000
: How large the jitter range should be to discourage synchronization. The timeout is randomized somewhat so as to prevent the possibility of several scriptroute servers implicitly self-synchronizing their traffic to a destination.

Operation. use_silk False
: Whether to use princeton silk modules. Silk is a set of kernel modules that provide "safe" raw socket access to application processes such as scriptroute that do not run as root. You want to say FALSE here. If you set this to True, see the Silk section below. Also, you may want to change Operation.syslog_facility.

Operation.service_port 3356
: What port to provide local service on. This is the port the interpreter connects to, not the port that provides remote service. An interpreter can be told to connect to a different port using –port, but the front-end interpreter cgi would need to be recompiled. This config option was added to work around some flaky kernels that would leave sockets in the listen state even when the application had terminated after causing a kernel oops.

Operation. pidfile /var/run/scriptrouted.pid
: A pidfile keeps track of the process id (pid) of the currently running scriptroute process. This simplifies init (rc.d) script writing. /var/run is the default, you may find pid files from other programs there. An empty pidfile implies not creating it. If you change this, you may also need to change the init script. (Or submit a patch that finds it automatically.)

Operation. assume_single_interface False
: Assume that the machine has only one interface. This means that the source address to be placed on packets is constant, and doesn't change by destination address. It is safe to leave this as False, but very likely that True will improve performance (reduce overhead) slightly without causing problems.

Operation.busy_wait_usec 500
: At the core of the scriptroute daemon is a select loop. Each packet to send is scheduled in a queue that is checked by select. When events are scheduled less than one kernel-scheduler quantum in the future, the daemon has to choose whether to sleep or spin. Spinning may allow the event to fire on time, at the cost of CPU cycles. We recommend that you leave this at zero if you're running a server out of the goodness of your heart. If you're engaged in hard core network measurement where precise packet spacing is important, frob this value up to a maximum of 2000 microseconds. (You can go higher, but there are diminishing returns).

4.1.3 Packetlog

Configuration of the sent packets log, used to provide a means to trace measurement traffic back to the source of the measurement.

Packetlog. file /var/log/scriptroute/packets

Where to log sent packets – doesn't use syslog, should be published via scriptroute web, possibly using a bind mount. If you stick this in a directory, remember to create the directory first. The log file will be owned by the user with uid -1 (nobody), though that may change to a getpwname() style lookup.

Packetlog.length_limit_Mbytes 100

How long to let the packet log file grow before no longer writing packets. An external program, like logrotate, should be used to rotate the logs; this is only a fail-safe to keep from filling a disk if lots of measurements are run.

Packetlog.bytes_per_packet 100

How many base64 encoded bytes of each packet should be logged - should be at least 80. The packetlog can fill up quickly, but holding on to at least a full header's worth of information is important. This number affects how much data after base 64 encoding is kept.

Packetlog.chomp_zeroes True

Whether to attempt to remove trailing zeroes from the packet when logging. This usually means dropping uninitialized data as an attempt to conserve disk space.

4.1.4 Pcap

The Pcap section allows customization of the parameters to libpcap functions. If you've found that you have to use the -i parameter with tcpdump, you'll want to modify the device settings. If you see too many "timeout without seeing sent packet" messages, you'll want to increase the buffer size

Pcap. buffer_resize True

Increase the size of the buffer used by libpcap. This allows the kernel to store more data in the socket before the scriptroute daemon fetches the data. The default is to try to increase the size of the buffer, though this may only have utility on linux.

Pcap.buffer_length 131072

If Pcap.buffer_resize is true, use this value in the call to setsockopt (SO_RCVBUF).

Pcap.autodetect_device True

Try to automatically discover which interfaces to connect to. If tcpdump on your system requires that you use the -i option to receive useful output (that is, tcpdump chooses the wrong interface to bind to) set this to false and edit hardwired_device below.

Pcap.hardwired_device eth0

If Pcap.autodetect_device is set above, bind to this named interface.

4.1.5 RateLimits

The RateLimits section consists of a set of token buckets that are used to limit the rate of outgoing traffic. Each of the token buckets has a capacity, a fill rate, and an initial level. Somewhat more intuitive, a capacity is a burst length: how much traffic can be generated at once. A fill rate represents the traffic rate that can be sustained over the course of a measurement. An initial level represents how full the bucket should be when first instantiated. Buckets are only destroyed after they are refilled.

RateLimits.dest.burst 8000
: Destination protection burst length. Each destination /24 gets a bucket. This is the throughput or byte-rate bucket. The default is large enough to send a few large packets in a burst, which is useful for packet-pair bottleneck bandwidth measurement.

RateLimits.dest.rate 1000
: Destination protection recharge rate. One can send 1 kB per second sustained to a destination.

RateLimits.dest. initial 4000
: Destination protection initial burst. The initial parameter is a bit unnecessary. It might as well be the same as burst.

RateLimits.destpkt.burst 10
: Destination protection burst length (packets). Each destination /24 gets a bucket. This is the packet rate bucket.

RateLimits.destpkt.rate 2
: Destination protection recharge rate (packets)

RateLimits.destpkt. initial 6
: Destination protection initial burst (packets)

RateLimits.syns.burst 5
: SYN-flood / port-scan rate limiting burst. Each measurement can source a limited number of TCP SYN packets at a limited rate.

RateLimits.syns.rate 1
: SYN-flood / port-scan rate limiting rate

RateLimits.syns. initial 2
: SYN-flood / port-scan rate limiting initial

RateLimits.source.burst 100000
: Source protection burst length (bytes)

RateLimits.source.rate 3000
: Source protection bytes per second sustained rate

RateLimits.source. initial 0
: Source protection burst initial burst length

RateLimits.disable False
: Disable rate limiting completely. This is highly unsafe and should not be attempted without supervision. If something goes wrong while rate limits are disabled, you‘re on your own. If true, Home.provide_service is forced FALSE.

RateLimits. disable_local True
: Disable rate limiting only for locally-run experiments. This somewhat unsafe and should not be used unless necessary. tulip expects to run without rate limits. If something goes wrong while rate limits are disabled, you‘re on your own.

RateLimits. destination_whitelist /etc/scriptroute/ratelimits_whitelist
: Disable destination rate limiting only for these IP addresses, intended to allow unbridled measurement of PlanetLab hosts.

4.1.6 ExperimentLimits

The ExperimentLimits section consists of the limits on the number of concurrent experiments the scriptroute daemon will support. While the scriptroute daemon could probably support an unlimited number of concurrent experiments, limiting them in number reduces the likelihood of interference between experiments, and limits the resources consumed by remote measurements.

ExperimentLimits.total_remote 10

How many experiments to run on behalf of remote clients at a time.

ExperimentLimits.total 50

How many experiments to run at a time, including local and remote.

ExperimentLimits.per_client 1

How many concurrent experiments to run per remote client (currently ignored - fixed at 1).

ExperimentLimits.packets_per_train_local 8000

How many packets can be placed in the argument to send train. This limit applies to experiments run on behalf of local users, which may be long-lived. Each packet is stored by the daemon so represents maintaining additional state.

ExperimentLimits.packets_per_train_remote 100

How many packets can be placed in the argument to send train. This limit applies to experiments run on behalf of remote users, which are already time limited. Packets may still be sent after the script that requested them runs out of time.

4.1.7 Home

The Home parameters control describe how to contact the dnskeeper that defines this system. Running your own dnskeeper would permit you to create a local scriptroute fiefdom that would not participate in the generally accessible system.

Home.server home.scriptroute.org.

Server running your DNS directory (new servers use home.scriptroute.org).

Home.port 3967

Port on which the DNS directory server runs (3967)

Home.provide_service True

Provide service to remote users (requires thttpd running). You can leave this set to true even without the webserver running. The dnskeeper running at Home.server will attempt to query your scriptroute server when it registers itself. If the query fails, either because you‘re behind a firewall or because your front-end thttpd isn‘t running, your server won‘t be registered anyway.

4.1.8 Silk

Pameters specific to using silk modules from Princeton. Unless you know you're running silk, these options are not useful.

Silk . udp_port 11234
: If using silk, what raw udp port? Will disappear if this can be allocated dynamically.

Silk . tcp_port 11234
: If using silk, what raw tcp port? Will disappear if this can be allocated dynamically.

Silk . icmp_sessionid 11234
: If using silk, what raw icmp id?

Silk . debug_unexpected_packets 0
: If using silk, packets we receive from the silk socket are very likely intended for the daemon. If the daemon doesn't agree, it could indicate a bug in the daemon logic or an unexpected corruption to packets. For example, if a middlebox modifies the IP identifier, the scriptroute daemon will be confused. This option prints to syslog the specified number of unexpected packets received. The number of packets to be printed is limited because the purpose is debugging, which doesn't need many packets, and logging received packets could exhaust disk space. Ignored if not using silk.

4.1.9 Filter

Filter parameters are used to configure both which packets can possibly be seen by the scriptroute daemon, and which packets can be sent as probes. There is both a filter installed for pcap and an array of filters to block outgoing traffic. The array is equivalent to a set of filters concatenated with 'or.' Any packet matching one of these filters is not sent. Only response packets matching the listen filter can be received

Filter . listen_filter ip and not ip dst 127.0.0.1 and not ip dst net 224.0.0.0/3 and not tcp port 22 and not udp port 2049 and ip[6:2] & 0x1fff = 0 and (icmp or not (ip src net 128.8.126.0/24 and ip dst net 128.8.126.0/24))
: Filter on what we sniff from the wire: mostly a performance optimization for ignoring local traffic. We also drop all multicast crud here to save processor time when multicast represents a decent fraction of traffic. We drop all port 2049 traffic and ip fragments (except the first) to avoid being overwhemled when NFS is in heavy use. This option is set by default_config to match the local subnet, and is quite important. Unused on Silk.

Filter . filter [0] not ip
: Source policy filters. Here we refuse to send anything bpf doesn't recognize as IP version 4. See tcpdump(1) for a discussion of bpf filters.

Filter . filter [1] not udp and not icmp and not tcp
: Here we disallow anything that is not udp, tcp, or icmp.

Filter . filter [2] ip broadcast or ip multicast or (ip dst net 0.0.0.127 mask 0.0.0.127 and not ip dst 128.114.63.127) or (ip dst net 0.0.0.0 mask 0.0.0.255)
: Here we disallow sending broadcast or multicast packets, intended to prevent smurf-style attacks. Broadcast packets are sent to destinations ending in .255, .127, or .0. Yes, there are others, but those subnets become successively smaller. You may choose to change the mask length to suit your level of paranoia.

Filter . filter [3] tcp and tcp[2:2] < 1024 and tcp[2:2] != 80
: TCP packets can be sent to high ports or port 80 only, to prevent port scanning or exploitation of common services.

Filter . filter [4] udp and (udp[2:2] < 1024 or udp[2:2] = 31337) and udp[2:2] != 53 and udp[2:2] != 123
: UDP to high ports or port 53 only. One can use scriptroute to send packets to name services. Recent network measurements (eg. King) have found utility in instrumenting recursive name lookups. Block contacting the default port Back Orifice, 31337.

Filter . filter [5] icmp and icmp[0] != 8 and icmp[0] != 13
: If we send an icmp packet, it must also be either echo request or timestamp request.

Filter . filter [6] ip[6:2] != 0 and ip[6] != 64

No fragments. The ‘don‘t fragment’ (df) bit is ok

Filter . filter [7] ip dst net 128.8.126.0/24 and not icmp

Additional filters may be added here, but please be careful when removing existing filters.

Filter . filter [8]

Filter . filter [9]

Filter . filter [10]

Filter . filter [11]

4.1.10 Credentials

The credentials module makes it easier to associate email addresses with clients. This association is made by scriptroute “cookies.‘’ This is useful when running scriptroute on resources that, while confident that nothing bad can happen, would like to know who is responsible for traffic if any problems occur and it becomes necessary to know what high-level experiment was run.

Credentials.require False

Do not execute measurements that lack valid credentials. This does not prevent local users from running scriptroute. The default is false to encourage servers to allow arbitrary users to connect.

Credentials.use True

Verify credentials if presented for the purpose of logging, but allow all submitted measurements to execute. Implied by Credentials.require. The default is true so that logs are as complete as possible

Credentials.accept [0]. id ns

Tag associated with my key. These tags should be short, and unique.

Credentials.accept [0]. key (public-key (rsa (n #00A14FC1CF44722383516A5AF392C888BA4139515688C6114E03290F8C5E9D7B87#) (e #010001#)))

My public key. Not my GPG public key, just the credentials signing key. It is in a gcrypt-formatted s-expression. See the auth subdirectory of scriptroute source for information on making your own cookie-generating key.

Credentials.accept [1]. id

More keys follow.

Credentials.accept [1]. key

Credentials.accept [2]. id

Credentials.accept [2]. key

Credentials.accept [3]. id

Credentials.accept [3]. key

Credentials.accept [4]. id

Credentials.accept[4].key

Credentials.accept[5].id

Credentials.accept[5].key

Credentials.accept[6].id

Credentials.accept[6].key

Credentials.accept[7].id

Credentials.accept[7].key

Credentials.accept[8].id

Credentials.accept[8].key

Credentials.accept[9].id

Credentials.accept[9].key

Credentials.accept[10].id

Credentials.accept[10].key

Credentials.accept[11].id

Credentials.accept[11].key

Chapter 5

Maintenance

Subscribe to scriptroute@cs.washington.edu. Releases will be announced there.

Alternately, if running a Debian system, be sure to update frequently.

Chapter 6

Conclusion

Wasn't this fun? I thought so.

More information at http://www.scriptroute.org.

For feedback, contact mailto:nspring@cs.washington.edu.

Bibliography

[1] J. Poskanzer. thttpd. http://www.acme.com/software/thttpd/.

[2] Simon. How to break out of a chroot() jail. http://www.bpfh.net/simes/computing/chroot-break.html, May 2002.

[3] D. Thomas and A. Hunt. *Programming Ruby: The Pragmatic Programmer's Guide*. Addison Wesley Longman, Inc., 2001. http://www.rubycentral.com/book/index.html.