Tcpreplay 3.x Manual (BETA)

Aaron Turner
http://tcpreplay.sourceforge.net/

4th May 2005

Overview

Tcpreplay is a suite of utilities for UNIX systems for editing and replaying network traffic which was previously captured by tools like tcpdump and ethereal. The goal of tcpreplay is to provide the means for providing reliable and repeatible means for testing a variety of network devices such as switches, router, firewalls, network intrusion detection and prevention systems (IDS and IPS).

Tcpreplay provides the ability to classify traffic as client or server, edit packets at layers 2-4 and replay the traffic at arbitrary speeds onto a network for sniffing or through a device.

Some of the advantages of using tcpreplay over using “exploit code” are:

- Since tcpreplay emulates the victim and the attacker, you generally only need a tcpreplay box and the device under test (DUT)
- Tests can include background traffic of entire networks without the cost and effort of setting up dozens of hosts or costly emulators
- No need to have a “victim” host which needs to have the appropriate software installed, properly configured and rebuilt after compromise
- Less chance that a virus or trojan might escape your network and wreak havoc on your systems
- Uses the open standard pcap file format for which dozens of command line and GUI utilities exist
- Tests are fully repeatable without a complex test harnesses or network configuration
- Tests can be replayed at arbitrary speeds
- Single command-line interface to learn and integrate into test harness
- You only need to audit tcpreplay, rather then each and every exploit individually
- Actively developed and supported by it’s author

Using this manual

The goal of this manual is to provide an idea of what tcpreplay and it’s utilities can do. It is not however intended to be a complete document which covers every possible use case or situation. It is also very much a work in progress and is far from complete and has numerous errors since a lot of things have changed since tcpreplay 2.x. It is expected that most of these issues will be ironed out before the offical 3.0 release is made. You should keep in mind the following conventions when reading this document:

- Commands you should run from the command line *are in italics*.
- Commands that should be run as root will have a ’#’ in front of them.
- Commands that should be run as an unprivelged user will have a ’$’ in front of them.
- Text that should be placed in a file `is in monospace.`

All of the applications shipped with tcpreplay support both short (a single dash followed by a single character) and long (two dashes followed by multiple characters) arguments. For consistancy, this document uses the long option format. Please review the man pages for the short argument equivalents.

Getting Help

If you still have a question after reading the Tcpreplay manual, man pages and FAQ, please contact the Tcpreplay-Users <tcpreplay-users@lists.sourceforge.net> mailing list. Note that if you ask a question which has clearly been covered in either the manual or FAQ, you will most likely be told to RTFM. Also, please try to explain your problem in detail. It is very difficult and fustrating to get requests from people seeking help only to have vague and incomplete information.

Corrections and additions to the manual

I've tried to keep this document up to date with the changes in tcpreplay, but occasionally I get too busy, make a mistake or just forget something. If you find anything in this document which could be improved upon, please let me know.

Getting Tcpreplay working on your system

Getting the source code

The source code is available as a tarball on the tcpreplay homepage: `http://tcpreplay.sourceforge.net/` I also encourage users familiar with Subversion to try checking out the latest code as it often has additional features and bugfixes not yet found in the offical releases.

```
$ svn checkout https://www.synfin.net:444/svn/tcpreplay/trunk tcpreplay
```

Requirements

1. Libnet[1] 1.1.x or better (1.1.3 fixes a checksum bug which effects tcprewrite)
2. Libpcap[2] 0.6.x or better (0.8.3 or better recommended)
3. To support the packet decoding feature you'll need tcpdump[3] binary installed.
4. You'll also need a compatible operating system. Basically, any *NIX operating system should work. Linux, *BSD, Solaris, OS X and others should all work. If you find any compatibility issues with any *NIX OS, please let me know.

Note: You will most likely experiance problems compiling tcpreplay if you have multiple copies of libnet and/or libpcap installed on your system. This generally is the case when your OS packaging system (such as RPM or dpkg) installs the libnet or libpcap library but not the header files (often installed as a seperate "dev" or "devel" package) and you also the library via source. To properly compile tcpreplay, you should **only have** one version of libpcap and libnet installed on your system.

Compiling Tcpreplay

Two easy steps:

1. $ *./configure && make*
2. # *make install*

There are some optional arguments which can be passed to the 'configure' script which may help in cases where your libnet, libpcap or tcpdump installation is not standard or if it can't determine the correct network interface card to use for testing. I also recommend that for beta code you specify **–enable-debug** to the configure script in case you find any bugs. If you find that configure isn't completing correctly, run: *./configure –help* for more information.

You may also choose to run: # *make test -i*

- make test is just a series of sanity checks which try to find serious bugs (crashes) in tcpprep and tcpreplay.

---

[1] http://www.packetfactory.net/libnet/

[2] http://www.tcpdump.org/

[3] http://www.tcpdump.org/

- make test requires at least one properly configured network interface. If the configure script can't guess what a valid interface is you can specify it with the –with-testnic and –with-testnic2 arguments.
- If make test fails, often you can find details in test/test.log.
- OpenBSD's make has a bug where it ignores the MAKEFLAGS variable in the Makefile, hence you'll probably want to run: *make -is test* instead.

Basic Tcpreplay Usage

Replaying the traffic

To replay a given pcap as it was captured all you need to do is specify the pcap file and the interface to send the traffic out interface 'eth0':

```
# tcpreplay –intf1=eth0 sample.pcap
```

Replaying at different speeds

You can also replay the traffic at different speeds then it was originally captured[4].

Some examples:

- To replay traffic as quickly as possible:
  ```
  # tcpreplay –topspeed –intf1=eth0 sample.pcap
  ```
- To replay traffic at a rate of 10Mbps:
  ```
  # tcpreplay –mbps=10.0 –intf1=eth0 sample.pcap
  ```
- To replay traffic 7.3 times as fast as it was captured:
  ```
  # tcpreplay –multiplier=7.3 –intf1=eth0 sample.pcap
  ```
- To replay traffic at half-speed:
  ```
  # tcpreplay –multiplier=0.5 –intf1=eth0 sample.pcap
  ```
- To replay at 25 packets per second:
  ```
  # tcpreplay –pps=25 –intf1=eth0 sample.pcap
  ```

Replaying files multiple times

Using the loop flag you can specify that a pcap file will be sent two or more times[5]:

- To replay the sample.pcap file 10 times:
  ```
  # tcpreplay –loop=10 –intf1=eth0 sample.pcap
  ```
- To replay the sample.pcap an infinitely or until CTRL-C is pressed:
  ```
  # tcpreplay –loop=0 –intf1=eth0 sample.pcap
  ```

Editing Packets

There are a number of ways you can edit packets stored in a pcap file:

1. Rewriting IP addresses so that they appear to be sent from and to different hosts
2. Fixing corrupted packets which were truncated by tcpdump or had bad checksums
3. Adding, removing or changing 802.1q VLAN tags on frames
4. Rewriting traffic so that it no longer uses "standard" TCP or UDP ports for the given service
5. Changing ethernet MAC addresses so that packets will be accepted by a router or firewall

---

[4] Tcpreplay makes a "best" effort to replay traffic at the given rate, but due to limitations in hardware or the pcap file itself, it may not be possible. Capture files with only a few packets in them are especially susceptible to inaccurately timing packets.

[5] Looping files resets internal counters which control the speed that the file is replayed. Also because the file has to be closed and re-opened, an added delay between the last and first packet may occur.

Splitting Traffic

Anything other then just replaying packets at different speeds requires additional work and CPU cycles. While older versions of tcpreplay allowed you to do many of these calculations while replaying traffic, it had a negative effect on the overall throughput and performance of tcpreplay. Hence, these secondary features have been placed in two utilities:

- tcpprep - Used to categorize packets as originating from clients or servers
- tcprewrite - Used to edit packets

By using tcpprep and tcprewrite on a pcap file before sending it using tcpreplay, many possibilities open up. A few of these possibilities are:

Classifying client and servers with tcpprep

Both tcpreplay and tcprewrite process a single pcap file and generate output. Some features, such as rewriting IP or MAC addresses or sending traffic out two different interfaces, require tcpreplay and tcprewrite to have some basic knowledge about which packets were sent by “clients” and “servers”. Such classification is often rather arbitrary since for example a SMTP mail server both accepts inbound email (acts as a server) and forwards mail to other mail servers (acts as a client). A webserver might accept inbound HTTP requests, but make client connections to a SQL server.

To deal with this problem, tcpreplay comes with tcpprep which provides a number of manual and automatic classification methods which cover a variety of situations.

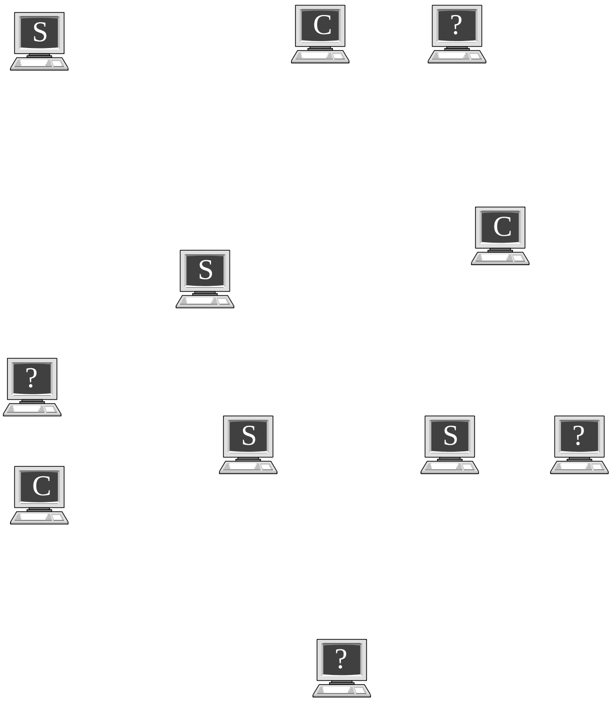
Seperating clients and servers automatically

The easiest way to split clients and servers is to let tcpprep do the classification for you. Tcpprep examines the pcap file for TCP three-way handshakes, DNS lookups and other types of traffic to figure out which IP’s mostly act like clients and which mostly act like servers. There are four different automatic modes that you can choose between:

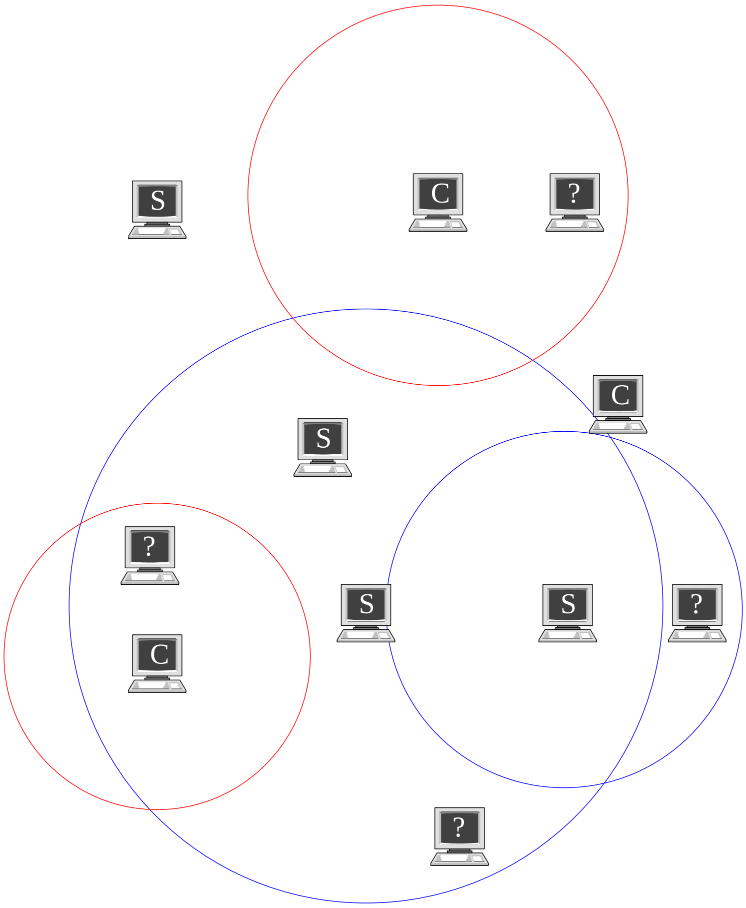
1. Bridge - This is the simplest mode. Each IP is individually tracked and ranked as a client or server. However, if any of the hosts do not generate enough “client” or “server” traffic then tcpprep will abort complaining that it was unable to determine its classification. This works best when clients and servers are intermixed on the same subnet.
2. Client - This works just like bridge mode, except that unknown hosts will be marked a client.
3. Server - This works just like bridge mode, except that unknown hosts will be marked a server.
4. Router - Hosts are first ranked as client or server. Then each host is placed in a subnet which is expanded until either all the unknown hosts are included or the –maxmask is reached. This works best when clients and servers are on diffierent networks.

TCPPREP AUTOMATIC ROUTER MODE PROCESS

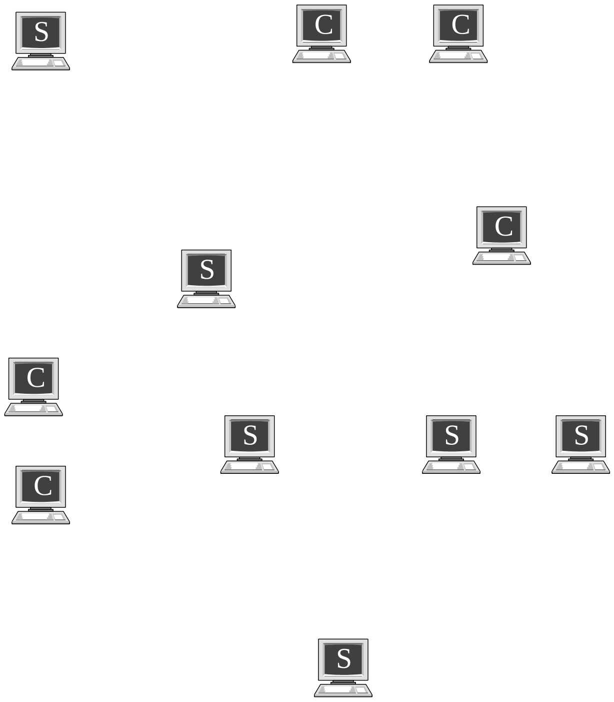
STEP 1: Categorize Clients, Servers and Unknowns



STEP 2: Clients and Servers Expand Their Subnets to Include Unknowns



STEP 3: Unknowns Now Marked as Clients and Servers



Classifying clients and servers in automatic mode is as easy as choosing a pcap file, an output “tcpprep cache file” and the mode to use:

```
$ tcpprep –auto=bridge –pcap=input.pcap –cachefile=input.cache
```

The above example would split traffic in bridge mode. Other modes are “router”, “client” and “server”. If you wish, you can override the default 2:1 ratio of server vs. client traffic required to classify an IP as a server. If for example you wanted to require 3.5 times as much server to client traffic you would specify it like:

```
$ tcpprep –auto=bridge –ratio=3.5 –pcap=input.pcap –cachefile=input.cache
```

Seperating clients and servers manually by subnet

Sometimes, you may not want to split traffic based on clients and servers. The alternative to using on of the automatic modes in this case, is to use one of the manual modes. One manual way of differentiating between clients and servers using tcpprep is by specifying a list of networks in CIDR notation which contain “servers”. Of course the specified CIDR netblocks don’t have to contain

Replaying on multiple interfaces

Tcpreplay can also split traffic so that each side of a connection is sent out a different interface[6]. In order to do this, tcpreplay needs the name of the second interface (-j) and a way to split the traffic. Currently, there are two ways to split traffic:

1. -C = split traffic by source IP address which is specified in CIDR notation
2. -c = split traffic according to a tcpprep cachefile[7]

When splitting traffic, it is important to remember that traffic that matches the filter is sent out the primary interface (–intf1). In this case, when splitting traffic by source IP address, you provide a list of networks in CIDR notation. For example:

- To send traffic from 10.0.0.0/8 out eth0 and everything else out eth1:
  ```
  tcpreplay -C 10.0.0.0/8 –intf1=eth0 –intf2=eth1 sample.pcap
  ```
- To send traffic from 10.1.0.0/24 and 10.2.0.0/20 out eth0 and everything else out eth1:
  ```
  tcpreplay -C 10.1.0.0/24,10.2.0.0/20 –intf1=eth0 –intf2=eth1 sample.pcap
  ```
- After using tcpprep to generate a cache file, you can use it to split traffic between two interfaces like this:
  ```
  tcpreplay -c sample.cache –intf1=eth0 –intf2=eth1 sample.pcap
  ```

Selectively sending or dropping packets

Sometimes, you want to do some post-capture filtering of packets. Tcpreplay let’s you have some control over which packets get sent.

1. -M = disables sending of martian packets. By definition, martian packets have a source IP of 0.x.x.x, 127.x.x.x, or 255.x.x.x
2. -x = send packets which match a specific pattern
3. -X = send packets which do not match a specific pattern

Both -x and -X support a variety of pattern matching types. These types are specified by a single character, followed by a colon, followed by the pattern. The following pattern matching types are available:

1. S - Source IP
   Pattern is a comma delimited CIDR notation
2. D - Destination IP
   Pattern is a comma delimited CIDR notation
3. B - Both source and destination IP must match
   Pattern is a comma delimited CIDR notation
4. E - Either source or destination IP must match
   Pattern is a comma delimited CIDR notation

---

[6] Note that you can also use the following options to split traffic into two files using -w and -W which are described later on in this FAQ.

[7] For information on generating tcpprep cache files, see the section on tcpprep.

5. P - A list of packet numbers from the pcap file.
   Pattern is a series of numbers, separated by commas or dashes.
6. F - BPF syntax (same as used in tcpdump).
   Filter must be quoted and is only supported with -x[8].

Examples:

- To only send traffic that is too and from a host in 10.0.0.0/8:
  ```
  tcpreplay -x B:10.0.0.0/8 –intf1 eth0 sample.pcap
  ```
- To not send traffic that is too or from a host in 10.0.0.0/8:
  ```
  tcpreplay -X E:10.0.0.0/8 –intf1 eth0 sample.pcap
  ```
- To send every packet except the first 10 packets:
  ```
  tcpreplay -X P:1-10 –intf1 eth0 sample.pcap
  ```
- To only send the first 50 packets followed by packets: 100, 150, 200 and 250:
  ```
  tcpreplay -x P:1-50,100,150,200,250 –intf1 eth0 sample.pcap
  ```
- To only send TCP packets from 10.0.0.1:
  ```
  tcpreplay -x F:’tcp and host 10.0.0.1’ –intf1 eth0 sample.pcap
  ```

Replaying only a few packets

Using the limit packets flag (-L) you can specify that tcpreplay will only send at most a specified number of packets.

- To send at most 100 packets:
  ```
  tcpreplay –intf1 eth0 -L 100 sample.pcap
  ```

Skipping the first bytes in a pcap file

If you want to skip the beginning of a pcap file, you can use the offset flag (-o) to skip a specified number of bytes and start sending on the next packet.

- To skip 15Kb into the pcap file and start sending packets from there:
  ```
  tcpreplay –intf1=eth0 -o 15000 sample.pcap
  ```

Replaying packets which are bigger then the MTU

Occasionally, you might find yourself trying to replay a pcap file which contains packets which are larger then the MTU for the sending interface. This might be due to the packets being captured on the loopback interface or on a 1000Mbps ethernet interface supporting “jumbo frames”. I’ve even seen packets which are 1500 bytes but contain both an ethernet header and trailer which bumps the total frame size to 1518 which is 4 bytes too large.

By default, tcpreplay will skip these packets and not send them. Alternatively, you can specify the -T flag to truncate these packets to the MTU and then send them. Of course this may invalidate your testing, but it has proven useful in certain situations. Also, when this feature is enabled, tcpreplay will automatically recalculate the IP and TCP, UDP or ICMP checksums as needed. Example:

```
tcpreplay –intf1 eth0 -T sample.pcap
```

Writing packets to a file

It’s not always necessary to write packets to the network. Since tcpreplay has so many features which modify and select which packets are sent, it is occasionally useful to save these changes to another pcap file for comparison. Rather then running a separate tcpdump process to capture the packets, tcpreplay now supports output directly to a file. Example:

```
tcpreplay –intf1 eth0 -w output.pcap -F -u pad -x E:10.0.0.0/8 input1.pcap input2.pcap input3.pcap
```

Notice that specifying an interface is still required (required for various internal functions), but all the packets will be written to *output.pcap*.

You can also split traffic into two files by using -W <2nd output file>.

---

[8] Note that if you want to send all the packets which do not match a bpf filter, all you have to do is negate the bpf filter. See the tcpdump(1) man page for more info.

Extracting Application Data (Layer 7)

New to version 2.0 is the ability to extract the application layer data from the packets and write them to a file. In the man page, we call this "data dump mode" which is enabled with -D. It's important to specify -D before -w (and -W if you're splitting data into two files). Example:

```
tcpreplay -D –intf1 eth0 -j eth0 -w clientdata -W serverdata -C 10.0.0.0/24 sample.pcap
```

Replaying Live Traffic

You can now replay live traffic sniffed on one network interface and replay it on another interface using the -S flag to indicate sniff mode and the appropriate snaplen in bytes (0 denotes the entire packet). You can also enabling bi-directional traffic using the bridge mode flag: -b.

NOTE: It is critical for your sanity (and to prevent your murder by your network administrators) that the input interface and the output interface be on separate networks and additionally that no other network devices (such as bridges, switches, routers, etc) be connecting the two networks, else you will surely get a networkstorm the likes that have not been seen for years.

- Send packets sniffed on eth0 out eth1:
  ```
  tcpreplay –intf1 eth1 -S 0 eth0
  ```
- Bridge two subnets connected to eth0 and eth1:
  ```
  tcpreplay –intf1 eth0 –intf2=eth1 -b -S 0
  ```

By default, tcpreplay listens in promiscuous mode on the specified interface, however if you only want to send unicasts directed for the local system and broadcasts, you can specify the "not_nosy" option in the configuration file or -n on the command line. Note that if another program has already placed the interface in promiscuous mode, the -n flag will have no effect, so you may want to use the -x or -X argument to limit packets.

Replaying Packet Capture Formats Other Than Libpcap

There are about as many different capture file formats as there are sniffers. In the interest of simplicity, tcpreplay only supports libpcap[9]. If you would like to replay a file in one of these multitude of formats, the excellent open source tool Ethereal easily allows you to convert it to libpcap. For instance, to convert a file in Sun's snoop format to libpcap, issue the command:

```
tethereal -r blah.snoop -w blah.pcap
```

and replay the resulting file.

Replaying Client Traffic to a Server

A common question on the tcpreplay-users list is how does one replay the client side of a connection back to a server. Unfortunately, tcpreplay doesn't support this right now. The major problem concerns syncing up TCP Seq/Ack numbers which will be different. ICMP also often contains IP header information which would need to be adjusted. About the only thing that could be easy to do is UDP, which isn't usually requested.

This is however a feature that we're looking into implementing in the flowreplay utility. If you're interested in helping work on this feature, please contact us and we'd be more then happy to work with you. At this time however, we don't have an ETA when this will be implemented, so don't bother asking.

Decoding Packets

If the tcpdump binary is installed on your system when tcpreplay is compiled, it will allow you to decode packets as they are sent without running tcpdump in a separate window or worrying about it capturing packets which weren't sent by tcpreplay.

- Decode packets as they are sent:
  ```
  tcpreplay –intf1 eth0 -v sample.pcap
  ```
- Decode packets with the link level header:
  ```
  tcpreplay –intf1 eth0 -v -A “-e” sample.pcap
  ```

---

[9] Note that some versions of tcpreplay prior to 1.4 also supported the Solaris snoop format.

- Fully decode and send one packet at a time:
  ```
  tcpreplay –intf1 eth0 -v -1 -A “-s0 -evvvxX” sample.pcap
  ```

Note that tcpreplay automatically applies the -n flag to disable DNS lookups which would slow down tcpdump too much to make it effective.

Packet Editing

Rewriting MAC addresses

If you ever want to send traffic to another device on a switched LAN, you may need to change the destination MAC address of the packets. Tcpreplay allows you to set the destination MAC for each interface independently using the -I and -J switches. As of version 2.1.0, you can also specify the source MAC via -k and -K. Example:

- To send traffic out eth0 with a destination MAC of your router (00:00:01:02:03:04) and the source MAC of the server (00:20:30:40:50:60):
  ```
  tcpreplay –intf1=eth0 -I 00:00:01:02:03:04 -k 00:20:30:40:50:60 sample.pcap
  ```
- To split traffic between internal (10.0.0.0/24) and external addresses and to send that traffic to the two interfaces of a firewall:
  ```
  tcpreplay –intf1=eth0 –intf2=eth1 -I 00:01:00:00:AA:01 -J 00:01:00:00:AA:02 -C 10.0.0.0/24 sample.pcap
  ```

Randomizing IP addresses

Occasionally, it is necessary to have tcpreplay rewrite the source and destination IP addresses, yet maintain the client/server relationship. Such a case might be having multiple copies of tcpreplay running at the same time using the same pcap file while trying to stress test firewall, IDS or other stateful device. If you didn’t change the source and destination IP addresses, the device under test would get confused since it would see multiple copies of the same connection occurring at the same time. In order to accomplish this, tcpreplay accepts a user specified seed which is used to generate pseudo-random IP addresses. Also, when this feature is enabled, tcpreplay will automatically recalculate the IP and TCP, UDP or ICMP checksums as needed. Example:

```
tcpreplay –intf1=eth0 -s 1239 sample.pcap &
tcpreplay –intf1=eth0 -s 76 sample.pcap &
tcpreplay –intf1=eth0 -s 239 sample.pcap &
tcpreplay –intf1=eth0 sample.pcap
```

Replaying (de)truncated packets

Occasionally, it is necessary to replay traffic which has been truncated by tcpdump. This occurs when the tcpdump snaplen is smaller then the actual packet size. Since this will create problems for devices which are expecting a full-sized packet or attempting checksum calculations, tcpreplay allows you to either pad the packet with zeros or reset the packet length in the headers to the actual packet size. In either case, the IP and TCP, UDP or ICMP checksums are recalculated. Examples:

- Pad truncated packets:
  ```
  tcpreplay –intf1=eth0 -u pad sample.pcap
  ```
- Rewrite packet header lengths to the actual packet size:
  ```
  tcpreplay –intf1=eth0 -u trunc sample.pcap
  ```

Rewriting Layer 2 with -2

Starting in the 2.0.x branch, tcpreplay can replace the existing layer 2 header with one of your choosing. This is useful for when you want to change the layer 2 header type or add a header for pcap files without one. Each pcap file tells the type of frame. Currently tcpreplay knows how to deal with the following pcap(3) frame types:

- DLT_EN10MB
  Replace existing 802.3/Ethernet II header
- DLT_RAW
  Frame has no Layer 2 header, so we can add one.

- DLT_LINUX_SLL
  Frame uses the Linux Cooked Socket header which is most commonly created with *tcpdump -i any* on a Linux system.

Tcpreplay accepts the new Layer 2 header as a string of comma separated hex values such as: 0xff,0xac,0x00,0x01,0xc0,0x64. Note that the leading ’0x’ is *not* required.

Potential uses for this are to add a layer 2 header for DLT_RAW captures or add/remove ethernet tags or QoS features.

Rewriting DLT_LINUX_SLL (Linux Cooked Socket) captures

Tcpdump uses a special frame type to store captures created with the “-i any” argument. This frame type uses a custom 16 byte layer 2 header which tracks which interface captured the packet and often the source MAC address of the original ethernet frame. Unfortunately, it never stores the destination MAC address and it doesn’t store a source MAC when the packet is captured on the loopback interface. Normally, tcpreplay can’t replay these pcap files because there isn’t enough information in the LINUX_SLL header to do so; however two options do exist:

1. You can send these packets with -2 which will replace the LINUX_SLL header with an ethernet header of your choosing.
2. You can specify a destination MAC via -I and -J in which case tcpreplay will use the stored source MAC and create a new 802.3 Ethernet header. Note that if the pcap contains loopback packets, you will also need to specify -k and/or -K to specify the source MAC as well or they will be skipped.

Rewriting IP Addresses (pseudo-NAT)

Pseudo-NAT allows the mapping of IP addresses in IPv4 and ARP packets from one subnet to another subnet of the same or different size. This allows some or all the traffic sent to appear to come from a different IP subnet then it actually was captured on.

The mapping is done through a user specified translation table comprised of one or more source and destination network(s) in the format of <srcnet>/<masklen>:<dstnet>/<masklen> deliminated by a comma. Mapping is done by matching IP addresses to the source subnet and rewriting the most significant bits with the destination subnet. For example:

```
tcpreplay –intf1=eth0 -N 10.100.0.0/16:172.16.10.0/24 sample.pcap
```

would match any IP in the 10.100.0.0/16 subnet and rewrite it as if it came from or sent to the 172.16.10.0/24 subnet. Ie: 10.100.5.88 would become 172.16.10.88 and 10.100.99.45 would become 172.16.10.45. But 10.150.7.44 would not be rewritten.

For any given IP address, the translation table is applied in order (so if there are multiple mappings, earlier maps take precedence) and occurs only once per IP (no risk of an address getting rewritten a second time).

Advanced pseudo-NAT

Pseudo-NAT also works with traffic splitting (using two interfaces or output files) but with a few important differences. First you have the option of specifying one or two pseudo-NAT tables. Using a single pseudo-NAT table means that the source and destination IP addresses of both interfaces are rewritten using the same rules. Using two pseudo-NAT tables (specifying -N <Table1> -N <Table2>) will cause the source and destination IP addresses to be rewritten differently for each interface using the following matrix:

	Out Primary Interface	Out Secondary Interface
Src IP	Table 1	Table 2
Dest IP	Table 2	Table 1

While seemingly a bit confusing, this feature provides a number of interesting possibilities such as the ability to rewrite the IP headers of packets in the case where traffic is captured on the loopback interface (and the source and destination address is always 127.0.0.1) so that tcpreplay can make it look like two different systems are talking to each other (you’ll probably also need to specify the source and destination MAC addresses via -I, -J, -k and -K).

IP Endpoints

While pseudo-NAT provides a great deal of flexibility, it is often more complicated then is necessary for testing of inline devices. As a simplier alternative, tcpreplay supports the concept of rewriting all traffic to so that it appears to be between two IP addresses:

```
tcpreplay –intf1=eth0 –intf2=eth1 -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

Will rewrite all the traffic so that it is between 10.0.0.1 and 10.1.1.1. The equivalent command using -N would be:

```
tcpreplay –intf1=eth0 –intf2=eth1 -c sample.cache -N 0.0.0.0/0:10.0.0.1 -N 0.0.0.0/0:10.1.1.1 sample.pcap
```

Unifying Dual-Outputs

Since a number of tcpreplay's packet editing functions require splitting traffic between client and servers, one problem that may arrise is needing to edit packets but still output to a single interface or file. The solution to this is to use the one output option -O which causes packets to be processed as if they will be split between the interfaces/files, but then always go out the primary interface or file. Note that even though only one interface/file will be written to, both -i and -j must be specified; although they can be the same physical interface.

```
tcpreplay –intf1=eth0 -j eth0 -O -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

Merging the output to a single file:

```
tcpreplay –intf1=eth0 -j eth0 -w rewrite.pcap -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

Tcpprep Usage

What is tcpprep?

Tcpreplay can send traffic out two network cards, however it requires the calculations be done in real-time. These calculations can be expensive and can significantly reduce the throughput of tcpreplay.

Tcpprep is a libpcap pre-processor for tcpreplay which enables using two network cards to send traffic without the performance hit of doing the calculations in real-time.

What are these 'modes' tcpprep has?

Tcpprep has three basic modes which require the user to specify how to split traffic.

- CIDR (-c) mode requires the user to provide a list of networks. Any packet with a source IP in one of these networks gets sent out the primary interface.
- Regex (-r) mode requires the user to provide a regular expression. Any packet with a source IP matching the regex gets sent out the primary interface.
- Port (-p) mode splits TCP/UDP traffic based on the destination port in the header. Normally, ports 0-1023 are considered "server" ports and everything else a client port. You can create your own custom mapping file in the same format as /etc/services (see the services(5) man page for details) by specifying -s <file>.

And four auto modes in which tcpprep decides how to split traffic. Auto modes are useful for when you don't know much about the contents of the dump file in question and you want to split traffic up based upon servers and clients.

- Auto/Router (-a -n router) mode trys to find the largest network(s) that contain all the servers and no clients. Any unknown system is automatically re-classified as servers if it's inside the server network(s), otherwise it is classified as a client.
- Auto/Bridge (-a -n bridge) mode makes the assumption that the clients and servers are horribly intermixed on the network and there's no way to subnet them. While this takes less processing time to create the cache file it is unable to deal with unknown systems.
- Auto/Client (-a -n client) mode which works just like Auto/Bridge mode, except that any system it can't figure out is treated like a client.
- Auto/Server (-a -n server) mode which works just like Auto/Bridge mode, except that any system it can't figure out is treated like a server.

Splitting traffic based upon IP address

Tcpprep supports the same CIDR mode that tcpreplay supports using the -c flag (tcpreplay uses -C). Additionally, tcpprep also supports regex(7) regular expressions to match source IP addresses using the -r flag.

Auto Mode

How does Auto/Bridge mode work?

Tcpprep does an initial pass over the libpcap file to build a binary tree (one node per IP). For each IP, it keeps track of how many times it was a client or server. It then does a second pass of the file using the data in the tree and the ratio to determine if an IP is a client or server. If tcpprep is unable to determine the type (client or server) for each and every packet, then auto/bridge mode will fail. In these cases, it is best to use a different auto mode.

How does Auto/Router mode work?

Tcpprep does the same first pass as Auto/Bridge mode. It then trys to convert the binary tree into a list of networks containing the servers. Finally it uses the CIDR mode with the list of server networks in a second pass of the libpcap file. Unlike auto/bridge mode, auto/router mode can always successfully split IP addresses into clients and servers.

Determining Clients and Servers

Tcpprep uses the following methods in auto/router and auto/bridge mode to determine if an IP address is a client or server:

- Client:
  - TCP with Syn flag set
  - UDP source/destination port 53 (DNS) without query flag set
  - ICMP port unreachable (destination IP of packet)
- Server:
  - TCP with Syn/Ack flag set
  - UDP source/destination port 53 (DNS) with query flag set
  - ICMP port unreachable (source IP of packet)

Client/Server ratio

Since a system may send traffic which would classify it as both a client and server, it's necessary to be able to weigh the traffic. This is done by specifying the client/server ratio (-R) which is by default set to 2.0. The ratio is the modifier to the number of client connections. Hence, by default, client connections are valued twice as high as server connections.

Selectively sending/dropping packets

Tcpprep supports the same -x and -X options to selectively send or drop packets.

Using tcpprep cache files with tcpreplay

Just run:

tcpreplay -c sample.cache –intf1=eth0 –intf2=eth1 sample.pcap

Commenting tcpprep cache files

In versions of tcpprep >= 2.1.0, you can specify a comment to be embeded in the tcpprep cache file. Comments are user specified and automatically include the command line arguments passed to tcpprep.

tcpprep -C "this is my comment" -i sample.pcap -o sample.cache <other args>

Or for no user comment, but still embed the command arguments:

tcpprep -C "" -i sample.pcap -o sample.cache <other args>

You can then later on print out the comments by running:

tcpprep -P sample.cache

Using Configuration Files

Each of the applications in the tcpreplay suite offers the choice of specifying configuration options in a config file in addition to the traditional command line. Each command line option has an equivalent config file option which is listed in the man page. To specify the configuration file you'd like to use, use the –load-opts=<filename> option.

Configuration files have one option per line, and lines beginning with the pound sign (#) are considered comments and ignored. An example config file follows:

————BEGIN CONFIG FILE————

```
# send traffic out 'eth0'
intf1 eth0

# loop 5 times
loop 5

# send traffic 2x as fast
multiplier 2
```

————END CONFIG FILE————

You would then execute:
# tcpreplay –load-opts=myconfigfile sample.pcap

You can also group configuration options for tcpprep, tcprewrite and tcpreplay in a single config file by placing section markers in the config file. An example:

————BEGIN CONFIG FILE————

```
cachefile=example.tcpprep

[TCPREPLAY]
intf1 eth0
intf2 eth1
topspeed

[TCPPREP]
auto=bridge
comment='This cache file was created with a config file'
pcap=sample.pcap

[TCPREWRITE]
infile=sample.pcap
outfile=newsample.pcap
vlan=add
vlan-tag=44
endpoints=10.0.0.1:10.0.1.1
```

————END CONFIG FILE————

Flowreplay Usage

While tcpreplay is a great way to test NIDS and firewalls, it can't be used to test servers or HIDS since tcpreplay can't connect to a service running on a device. The solution to this problem is flowreplay which instead of sending packets at Layer 2 (ethernet header and up), it can actually connect via TCP or UDP to server and then sends and receives data based upon a pcap capture file created with a tool like Ethereal or tcpdump.

Please note that flowreplay is currently alpha quality and is missing a number of key features.

How flowreplay works

Put simply, flowreplay opens a socket connection to a service on a target system(s) and sends data over that socket based on the packet capture. Flowreplay has no understanding of the application protocol (like HTTP or FTP) so it is somewhat limited in how it can deal with complicated exchanges between client and server.

Some of these limitations are:

- Flowreplay only plays the client side[10] of the connection.
- Flowreplay doesn't understand the application protocols. Hence it can't always deal with the case when the server sends a different response then what was originally captured in the pcap file.
- Flowreplay only sends TCP and UDP traffic.
- Flowreplay doesn't know about multi-flow protocols like FTP.
- Flowreplay can't listen on a port and wait for a client to connect to it.

Running flowreplay

See the flowreplay(8) man page for details.

Tuning OS's for high performance

Regardless of the size of physical memory, UNIX kernels will only allocate a static amount for network buffers. This includes packets sent via the "raw" interface, like with tcpreplay. Most kernels will allow you to tweak the size of these buffers, drastically increasing performance and accuracy.

NOTE: The following information is provided based upon our own experiences or the reported experiences of others. Depending on your hardware and specific hardware, it may or may not work for you. It may even make your system horribly unstable, corrupt your harddrive, or worse.

NOTE: Different operating systems, network card drivers, and even hardware can have an effect on the accuracy of packet timestamps that tcpdump or other capture utilities generate. And as you know: garbage in, garbage out.

NOTE: If you have information on tuning the kernel of an operating system not listed here, please send it to me so I can include it.

Linux 2.4.x

The following is known to apply to the 2.4.x series of kernels. If anyone has any information regarding other kernel versions, please let us know. By default Linux's tcpreplay performance isn't all that stellar. However, with a simple tweak, relatively decent performance can be had on the right hardware. By default, Linux specifies a 64K buffer for sending packets. Increasing this buffer to about half a megabyte does a good job:

```
echo 524287 >/proc/sys/net/core/wmem_default
echo 524287 >/proc/sys/net/core/wmem_max
echo 524287 >/proc/sys/net/core/rmem_max
echo 524287 >/proc/sys/net/core/rmem_default
```

On one system, we've seen a jump from 23.02 megabits/sec (5560 packets/sec) to 220.30 megabits/sec (53212 packets/sec) which is nearly a 10x increase in performance. Depending on your system and capture file, different numbers may provide different results.

*BSD

*BSD systems typically allow you to specify the size of network buffers with the NMBCLUSTERS option in the kernel config file. Experiment with different sizes to see which yields the best performance. See the options(4) man page for more details.

Understanding Common Error and Warning Messages

Libnet can't open eth0: libnet_select_device(): Can't find interface eth0

Generally this occurs when the interface (eth0 in this example) is not up or doesn't have an IP address assigned to it.

---

[10] Flowreplay assumes the first UDP packet on a given 4-tuple is the client

Can’t open lo: libnet_select_device(): Can’t find interface lo

Version 1.1.0 of Libnet is unable to send traffic on the loopback device. Upgrade to a later release of the Libnet library to solve this problem.

Libnet can’t open eth0: libnet_init(): UID or EUID of 0 required

Tcpreplay requires that you run it as root.

100000 write attempts failed from full buffers and were repeated

When tcpreplay displays a message like "100000 write attempts failed from full buffers and were repeated", this usually means the kernel buffers were full and it had to wait until memory was available. This is quite common when replaying files as fast as possible with the "-R" option. See the tuning OS section in this document for suggestions on solving this problem.

Unable to process test.cache: cache file version missmatch

Cache files generated by tcpprep and read by tcpreplay are versioned to allow enhancements to the cache file format. Anytime the cache file format changes, the version is incremented. Since this occurs on a very rare basis, this is generally not an issue; however anytime there is a change, it breaks compatibility with previously created cache files. The solution for this problem is to use the same version of tcpreplay and tcpprep to read/write the cache files. Cache file versions match the following versions of tcpprep/tcpreplay:

- Version 1:
  Prior to 1.3.beta1
- Version 2:
  1.3.beta2 to 1.3.1/1.4.beta1
- Version 3:
  1.3.2/1.4.beta2 to 2.0.3
- Version 4:
  2.1.0 and above. Note that prior to version 2.3.0, tcpprep had a bug which broke cache file compatibility between big and little endian systems.

Skipping SLL loopback packet.

Your capture file was created on Linux with the ’any’ parameter which then captured a packet on the loopback interface. However, tcpreplay doesn’t have enough information to actual send the packet, so it skips it. Specifying a source and destination MAC address (-I, -k, -J, -K) will allow tcpreplay to send these packets.

Packet length (8892) is greater then MTU; skipping packet.

The packet length (in this case 8892 bytes) is greater then the maximum transmition unit (MTU) on the outgoing interface. Tcpreplay must skip the packet. Alternatively, you can specify the -T option and tcpreplay will truncate the packet to the MTU size, fix the checksums and send it.

Why is tcpreplay not sending all the packets?

Every now and then, someone emails the tcpreplay-users list, asking if there is a bug in tcpreplay which causes it not to send all the packets. This usually happens when the user uses the –topspeed flag or is replaying a high-speed pcap file (> 50Mbps, although this number is dependant on the hardware in use).

The short version of the answer is: no, we are not aware of any bugs which might cause a few packets to not be sent.

The longer version goes something like this:

If you are running tcpreplay multiple times and are using tcpdump or other packet sniffer to count the number packets sent and are getting different numbers, it’s not tcpreplay’s fault. The problem lies in one of two places:

1. It is well known that tcpdump and other sniffers have a problem keeping up with high-speed traffic. Furthermore, the OS in many cases *lies* about how many packets were dropped. Tcpdump will repeat this lie to you. In other words, tcpdump isn't seeing all the packets. Usually this is a problem with the network card or driver which may or may not be fixable. Try another network card/driver.
2. When tcpreplay sends a packet, it actually gets copied to a send buffer in the kernel. If this buffer is full, the kernel is supposed to tell tcpreplay that it didn't copy the packet to this buffer. If the kernel has a bug which squelches this error, tcpreplay will not keep trying to send the packet and will move on to the next one. Currently I am not aware of any OS kernels with this bug, but it is possible that it exists. If you find out that your OS has this problem, please let me know so I can list it here.

If for some reason, you still think its a bug in tcpreplay, by all means read the code and tell me how stupid I am. The do_packets() function in do_packets.c is where tcpreplay processes the pcap file and sends all of the packets.

Required Libraries and Tools

Libpcap

As of tcpreplay v1.4, you'll need to have libpcap installed on your system. As of v2.0, you'll need at least version 0.6.0 or better, but I only test our code with the latest version. Libpcap can be obtained on the tcpdump homepage[11].

Libnet

Tcpreplay v1.3 is the last version to support the old libnet API (everything before 1.1.x). As of v1.4 you will need to use Libnet 1.1.0 or better which can be obtained from the Libnet homepage[12].

Tcpdump

As of 2.0, tcpreplay uses tcpdump (the binary, not code) to decode packets to STDOUT in a human readable (with practice) format as it sends them. If you would like this feature, tcpdump must be installed on your system.

NOTE: The location of the tcpdump binary is hardcoded in tcpreplay at compile time. If tcpdump gets renamed or moved, the feature will become disabled.

Other Resources

Other pcap tools available

Tools to capture network traffic or decode pcap files

- tcpdump
  `http://www.tcpdump.org/`
- ethereal
  `http://www.ethereal.com/`
- ettercap
  `http://ettercap.sourceforge.net/`

---

[11] `http://www.tcpdump.org/`

[12] `http://www.packetfactory.net/Projects/Libnet/`

Tools to edit pcap files

- tcpslice
  Splits pcap files into smaller files
  `http://www.tcpdump.org/`

- mergecap
  Merges two pcap capture files into one
  `http://www.ethereal.com/`

- editcap
  Converts capture file formats (pcap, snoop, etc)
  `http://www.ethereal.com/`

- netdude
  GTK based pcap capture file editor. Allows editing most anything in the packet.
  `http://netdude.sourceforge.net/`

Other useful tools

- capinfo
  Prints statistics and basic information about a pcap file
  `http://www.ethereal.com/`

- text2pcap
  Generates a pcap capture file from a hex dump
  `http://www.ethereal.com/`

- tcpflow
  Extracts and reassembles the data portion on a per-flow basis on live traffic or pcap capture files
  `http://www.circlemud.org/~jelson/software/tcpflow/`