Tcpreplay 3.x FAQ

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

July 17, 2006

Contents

1 General Info

1.1 What is this FAQ for?

Tcpreplay is a suite of powerful tools, but with that power comes complexity. While I have done my best to write good man pages for tcpreplay and it's associated utilities, I understand that many people may want more information then I can provide in the man pages. Additionally, this FAQ attempts to cover material which I feel will be of use to people using tcpreplay, as well as common questions that occur on the Tcpreplay-Users <tcpreplay-users@lists.sourceforge.net> mailing list.

1.2 What tools come with tcpreplay?

- tcpreplay - replay ethernet packets stored in a pcap file as they were captured
- tcprewrite - edit packets stored in a pcap file
- tcpprep - a pcap pre-processor for tcpreplay
- flowreplay[1] - connects to a server(s) and replays the client side of the connection stored in a pcap file

1.3 What tools no longer come with Tcpreplay?

Recently, other people and projects have developed better versions of two applications that shipped with tcpreplay 2.x:

- pcapmerge - merges two or more pcap files into one. Ethereal now ships with a more powerful appliation called 'mergecap'.
- capinfo - displays basic information about a pcap file. Ethereal now ships with a more powerful application of the same name.

1.4 How can I get tcpreplay's source?

The source code is available in tarball format 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 found in the tarballs.

svn checkout https://www.synfin.net/svn/tcpreplay/trunk tcpreplay

1.5 What requirements does tcpreplay have?

1. You'll need recent versions of the libnet[2] and libpcap[3] libraries.
2. To support the packet decoding feature you'll need tcpdump[4] installed.
3. You'll also need a compatible operating system. Basically, any UNIX-like or UNIX-based operating system should work. Linux, *BSD, Solaris, OS X and others should all work. If you find any compatibility issues with any UNIX-like/based OS, please let me know.

1.6 Are there binaries available?

The tcpreplay project does not maintain binaries for any platforms. However some operating systems such as Debian GNU/Linux (apt-get) and OS X (fink) have packages available. Try searching on Google.

1.7 Is there a Microsoft Windows port?

Not really. We had one user port the code over for an old version of tcpreplay to Windows. Now we're looking for someone to help merge and maintain the code in to the main development tree. If you're interested in helping with this please contact Aaron Turner or the tcpreplay-users list. Other then that, you can download the tcpreplay-win32.zip file from the website and give it a go. Please understand that the Win32 port of tcpreplay comes with no support whatsoever, so if you run into a problem you're on your own.

[1] Flowreplay is still "alpha" quality and is not usable for most situations. Anyone interested in helping me develop flowreplay is encouraged to contact me.
[2] http://www.packetfactory.net/libnet/
[3] http://www.tcpdump.org/
[4] http://www.tcpdump.org/

1.8 How is tcpreplay licensed?

Tcpreplay is licensed under a three clause BSD-style license. For details see the docs/LICENSE file included with the source code.

1.9 What is tcpreplay?

In the simplest terms, tcpreplay is a tool to send network traffic stored in pcap format back onto the network; basically the exact opposite of tcpdump. Just to make things more confusing, tcpreplay is also a suite of tools: tcpreplay, tcpprep, tcprewrite and flowreplay.

1.10 What are some uses for tcpreplay?

Originally, tcpreplay was written to test network intrusion detection systems (NIDS), however tcpreplay has been used to test firewalls, routers, and other network devices. With the addition of flowreplay, most[5] any udp or tcp service on a server can be tested as well.

1.11 What are some uses for flowreplay?

A lot of people wanted a tool like tcpreplay, but wanted to be able to replay traffic *to* a server. Since tcpreplay was unable to do this, I developed flowreplay which replays the data portion of the flow, but recreates the connection to the specified server(s). This makes flowreplay an ideal tool to test host intrusion detection systems (HIDS) as well as captured exploits and security patches when the actual exploit code is not available. Please note that flowreplay is still alpha quality code which means it doesn't work very well (some would argue it doesn't work at all) and is currently missing some important features. Feel free to try flowreplay, but unless you're willing and able to contribute, don't bother complaining that it doesn't work.

1.12 What is the history of tcpreplay?

Tcpreplay has had quite a few authors over the past five or so years. One of the advantages of the BSD and GPL licenses is that if someone becomes unable or unwilling to continue development, anyone else can take over.

Originally, Matt Undy of Anzen Computing wrote tcpreplay. Matt released version 1.0.1 sometime in 1999. Sometime after that, Anzen Computing was (at least partially) purchased by NFR and development ceased.

Then in 2001, two people independently started work on tcpreplay: Matt Bing of NFR and Aaron Turner of OneSecure. After developing a series of patches (the -adt branch), Aaron attempted to send the patches in to be included in the main development tree.

After some discussion between Aaron and Matt Bing, they decided to continue development together. Since then, two major rewrites have occured, and more then thirty new features have been added, including the addition of a number of accessory tools.

Today, Aaron continues active development of the code.

2 Bugs, Feature Requests, and Patches

2.1 Where can I get help, report bugs or contact the developers?

The best place to get help or report a bug is the Tcpreplay-Users mailing list:
`http://lists.sourceforge.net/lists/listinfo/tcpreplay-users`

Please do not email the author directly as it prevents others from learning from your questions.

2.2 What information should I provide when I report a bug?

One of the most frustrating things for any developer trying to help a user with a problem is not enough information. Please be sure to include *at minimum* the following information, however any additional information you feel may be helpful will be appreciated.

- Version information (output of -V)

[5] Note the flowreplay does not support protocols such as ftp which use multiple connections.

- Command line used (options and arguments)
- Platform (Red Hat Linux 9 on Intel, Solaris 7 on SPARC, etc)
- Error message (if available) and/or description of problem
- If possible, attach the pcap file used (compressed with bzip2 or gzip preferred)
- The core dump or backtrace if available

2.3 I have a feature request, what should I do?

Let us know! Many of the features exist today because users like you asked for them. To make a feature request, email the tcpreplay-users mailing list (see above).

2.4 I've written a patch for tcpreplay, how can I submit it?

I'm always willing to include new features or bug fixes submitted by users. You may email me directly or the tcpreplay-users mailing list. Please *do not* use the Patch Tracker on the tcpreplay SourceForge web site. But before you start working on adding a feature or fixing a bug in tcpreplay, please make sure you checkout the latest source code from the Subversion repository. Patches against released versions are almost surely not going to apply cleanly if at all.

2.5 Patch requirements

- Be aware that submitting a patch, *you are assigning your copyright to me.* If this is not acceptable to you, then *do not* send me the patch! I have people assign their copyright to me to help prevent licensing issues that may crop up in the future.
- Please provide a description of what your patch does!
- Comment your code! I won't use code I can't understand.
- Make sure you are patching a branch that is still being maintained. Generally that means that most recent stable and development branches (2.0 and 3.0 at the time of this writing).
- Make sure you are patching against the most recent release for that branch.
- Please submit your patch in the *unified diff* format so I can better understand what you're changing.
- Please provide any relevant personal information you'd like listed in the CREDITS file.

Please note that while I'm always interested in patches, I may rewrite some or all of your submission to maintain a consistent coding style.

3 Understanding tcpprep

3.1 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.

3.2 How does tcpprep work?

Tcpprep reads in a libpcap (tcpdump) formatted capture file and does some processing to generate a tcpreplay cache file. This cache file tells tcpreplay which interface a given packet should be sent out of.

3.3 Does tcpprep modify my libpcap file?

No.

3.4 Why use tcpprep?

There are three major reasons to use tcpprep:

1. Tcpprep can split traffic based upon more methods and criteria then tcpreplay.
2. By pre-processing the pcap, tcpreplay has a higher theoretical maximum throughput.
3. By pre-processing the pcap, tcpreplay can be more accurate in timing when replaying traffic at normal speed.

3.5 Can a cache file be used for multiple (different) libpcap files?

Cache files have nothing linking them to a given libpcap file, so there is nothing to stop you from doing this. However running tcpreplay with a cache file from a different libpcap source file is likely to cause a lot of problems and is not supported.

3.6 Why would I want to use tcpreplay with two network cards?

Tcpreplay traditionally is good for putting traffic on a given network, often used to test a network intrusion detection system (NIDS). However, there are cases where putting traffic onto a subnet in this manner is not good enough- you have to be able to send traffic *through* a device such as a IPS, router, firewall, or bridge.

In these cases, being able to use a single source file (libpcap) for both ends of the connection solves this problem.

3.7 How big are the cache files?

Very small. Actual size depends on the number of packets in the dump file. Two bits of data is stored for each packet. On a test using a 900MB dump file containing over 500,000 packets, the cache file was only 150K.

4 Common Error and Warning Messages

4.1 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.

4.2 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.

4.3 Can't open eth0: UID != 0

Tcpreplay requires that you run it as root.

4.4 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.

4.5 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.

4.6 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 destination and source MAC address (-D and -S) will allow tcpreplay to send these packets.

4.7 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. This often occurs with pcaps captured over loopback interfaces which have much larger MTU's then ethernet.

5 Common Questions from Users

5.1 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 -t 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, driver or OS kernel 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.

5.2 Can tcpreplay read gzip/bzip2 compressed files?

Yes, but not directly. Since tcpreplay can read data via STDIN, you can decompress the file on the fly like this:

gzcat myfile.pcap.gz | tcpreplay -i eth0 -

Note that decompressing on the fly will require additional CPU time and will likely reduce the overall performance of tcpreplay.

5.3 How fast can tcpreplay send packets?

First, if performance is important to you, then upgrading to tcpreplay 3.x is worthwhile since it is more optimized then the 1.x or 2.x series. After that, there are a number of variables which effect performance, including on how you measure it (packets/sec or bytes/sec). 100Mbps and 120K pps are quite doable. Generally speaking here are some points to consider:

- Profiling tcpreplay has shown that a significant amount of time is spent writing packets to the network. Hence, your OS kernel implimentation of writing to raw sockets is one of the most important aspects since that is where tcpreplay spends most of it's time.
- Like most network based I/O, it is faster to send the same amount of data in a few large packets then many small packets.
- Most operating systems will cache disk reads in RAM; hence making subsequent access to the file faster the second time.
- Re-opening small files repeatly will reduce performance. Consider using mergecap to generate a single large file.
- Network cards and drivers, disk speed (RPM is more important then seek), amount of RAM and system bus speed are all important.
- In general servers with faster disks and bus speeds will be faster then desktops which will be faster then laptops.

5.4 Is tcpreplay stateful?

No. Tcpreplay processes each packet in the order it is stored in the pcap file. The default is to send each packet based on the timestamp stored in the pcap file. If your pcap file has packets out of order, tcpreplay will send them out of order. In certain situations a packet may have an earlier timestamp then the packet before it, tcpreplay will then send the second packet as soon as possible.

The basic point is that if your pcap file is well formed and has the packets in the correct order, then tcpreplay will create a "stateful" packet stream. If your pcap file has errors, then tcpreplay will repeat those errors. Garbage in, garbage out.

6 Testing Methodologies

A topic which comes up regularly, is how to use tcpreplay to test products like intrusion detection/prevention devices (IDS/IPS) and deep inspection firewalls. Generally, I hear people suggest three things:

1. Use security scanners like Nessus
2. Use "real attacks" like those generated by Metasploit
3. Use a replay tool like tcpreplay to generate attack traffic

First, let me say that security scanners like Nessus do a really crappy job of testing the effectiveness of IDS/IPS and firewalls. The simple reason is that security scanners don't try to exploit vulnerabilities because it creates problems on the network. IT managers don't like it when their servers start rebooting or routers crash, so scanners use other non-agressive techniques like banner grabbing to find potentially vulnerable systems. Simply put, these non-agressive techniques often look nothing like a real attack.

That leaves generating "real attacks" and replay tools.

Advantages of real attacks:

- It's clear when you have a valid test case because the target system is compromised
- Exploit code and attack tools are widely available for many attacks

Disadvantages of real attacks:

- After the test case is run, the target system may be unstable or corrupted, requiring a reboot or re-install
- Generally requires two systems: a target (often running VMWare) and an attacker system
- Installing, configuring and managing various operating systems and applications to attack is a lot of work
- Difficult to automate test cases since there is no standardized interface to these tools
- You have to be careful about trojaned exploit code or worms which escape your lab

Advantages of replay tools:

- Since both the victim and attacker are virtual, there is no need to reboot/re-install systems after each test
- A complete test bed requires only a single system with two NIC's
- Once you have a library of pcap files, there is virtually zero management overhead
- Replay tools provide a common interface to emulating any attack against any OS/application making automation simple
- Pcap files are not executable, so trojans and escaping worms aren't an issue

Disadvantages of replay tools;

- There are trust issues regarding pcap files. Are you 100% sure that pcap file is correct (not corrupted, doesn't have truncated packets, actually contains the valid exploit)
- There are few publicly available pcap's which contain attacks useful for testing so you must create your own

7 Required Libraries and Tools

7.1 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[6].

7.2 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[7].

7.3 Libpcapnav

Starting with v2.0, tcpreplay can use libpcapnav to support the jump offset feature. If libpcapnav is not found on the system, that feature will be disabled. Libpcapnav can be found on the NetDude homepage[8].

7.4 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.

[6] http://www.tcpdump.org/
[7] http://www.packetfactory.net/Projects/Libnet/
[8] http://netdude.sourceforge.net/

8 Other pcap tools available

8.1 Tools to capture network traffic or decode pcap files

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

8.2 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.ethreal.com/`
- pcapmerge
  Merges two or more pcap capture files into one
  `http://tcpreplay.sourceforge.net/`
- editcap
  Converts capture file formats (pcap, snoop, etc)
  `http://www.ethreal.com/`
- netdude
  GTK based pcap capture file editor. Allows editing most anything in the packet.
  `http://netdude.sourceforge.net/`

8.3 Other useful tools

- capinfo
  Prints statistics and basic information about a pcap file
  `http://tcpreplay.sourceforge.net/`
- text2pcap
  Generates a pcap capture file from a hex dump
  `http://www.ethreal.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/`