TCPCopy Manual

NetEase, Inc.

2013.09.18

Table of Contents

1 Overview

TCPCopy is a TCP stream replay tool mainly developed by NetEase and also contributed by developers in other companies (e.g., Taobao). It has been widely used in Internet companies in China, including NetEase, Taobao, Sina, Sohu, etc.

TCPCopy could copy the TCP live flow on the online server to a target server, and thus bring the complexity of online environments to the target server. It is especially useful for live testing and reducing errors before a system being deployed online. For example, when you want to migrate from Apache to Nginx, TCPCopy can help you test it. While Apache is running online, TCPCopy can copy the TCP flows from Apache to Nginx. To Nginx, the TCP flows are just forwarding to it. This will not affect Apache at all except cost a little network bandwidth and CPU load.

TCPCopy could be used in the following areas.

1) Distributed stress testing

Copy real-world data to stress test your server software. Bugs that only can be produced in high-stress situations can be found.

2) Live testing

Prove the new system is stable and find bugs that only occur in the real world.

3) Benchmark

Do performance benchmark. For instance, you can use TCPCopy to compare the performance of Apache and Nginx.

4) Regression testing

5)...

2 How Does TCPCopy Work?

Generally speaking, TCPCopy copies packets on the online server, modifies TCP/IP headers, and sends modified packets to the target server. In this way, TCP applications on the target server will consider the packets from the online server as online requests from end-users. As most applications on the Internet are based on the TCP protocol, TCPCopy could be widely used for applications based on HTTP, memcached, POP3, mysql, etc.

There are two ways to use TCPCopy: adopting the traditional architecture or using the advanced architecture.

2.1 The traditional architecture

The traditional architecture of TCPCopy can be seen in Figure 1.

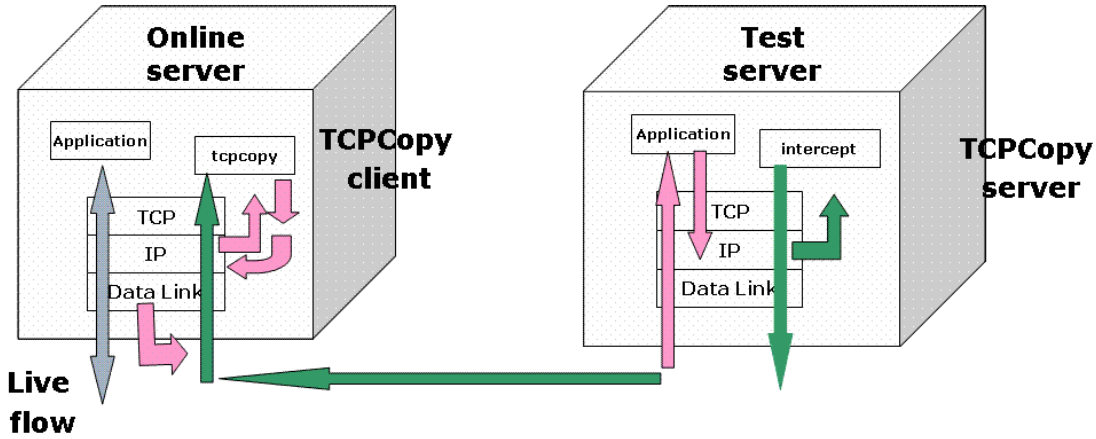

Figure 1. Traditional architecture of TCPCopy

As shown in Figure 1, TCPCopy consists of two parts: the TCPCopy client (***tcpcopy***) and the TCPCopy server (***intercept***). While the TCPCopy client runs on the online server and captures the online requests, the TCPCopy server runs on the test server and does some assistant work, such as passing response info to the TCPCopy client and filtering outbound traffic.

The TCPCopy client (*tcpcopy*) utilizes *raw socket input* technique by default to capture the online packets at the network layer and does the necessary processing (including TCP interaction simulation, network latency control, and common upper-layer interaction simulation), and uses *raw socket output* technique by default to send packets to the test server.

The TCPCopy server (*intercept*) is responsible for passing the response header to the TCPCopy client. By setting the *iptables* command, locally generated response packets will be sent to the corresponding kernel module (*ip_queue* or *nfqueue*), and then the kernel module will attempt to deliver the packets to the TCPCopy server (*intercept*), which will extract response header information and determine whether to drop the packet or not. To make the TCPCopy client send the next packet, the TCPCopy server (*intercept*) often needs to send the response header to the TCPCopy client using a special channel. When the TCPCopy client(*tcpcopy*) receives the response header, it utilizes the header information to modify the attributes of online packets and continues to send another packet.

It should be noticed that the responses from the test server are dropped at the network layer of the test server and not return to the end-user by default. There are two reasons to do this.

1) It will not burden the output bandwidth, and can save money.

2) It will not influence the TCP/IP protocol of the end-user.

2.2 The advanced architecture

The difference between the advanced architecture and the traditional architecture is that the TCPCopy server (*intercept*) runs on a separate machine instead of the test server. Thus, the test tasks will not be influenced by the TCPCopy server (*intercept*).

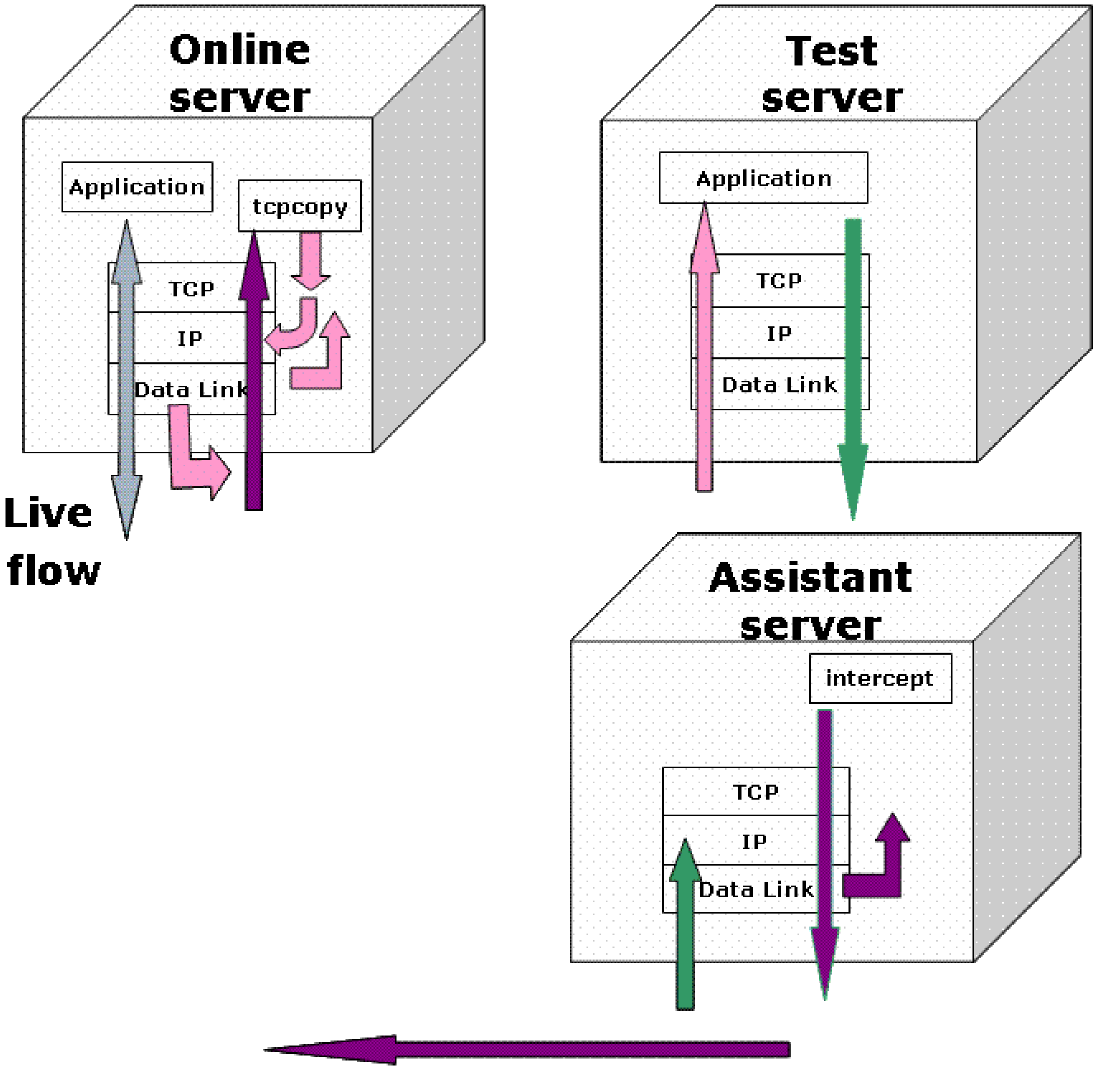

Figure 2. Advanced architecture of TCPCopy

The advanced architecture of TCPCopy can be seen in Figure 2. Assume the online server is running online services, the test server is used to do the test tasks and the assistant server is adopted to run the TCPCopy server (*intercept*). The only operation needed in the test server for TCPCopy is setting appropriate route commands to route response packets to the assistant server. The TCPCopy server (*intercept*) at the assistant server captures response packets at the data link layer and passes the response header to the TCPCopy client on the online server. These changes lead to more realistic testing because the test task in the test server is no longer influenced by the TCPCopy server (*intercept*). Moreover, as the TCPCopy server (*intercept*) captures packets more efficiently at the data link layer and multiple instances of the TCPCopy server (*intercept*) could run concurrently, the processing ability of the TCPCopy server (*intercept*) is also enhanced.

3 TCPCopy vs Tcpreplay

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/ wireshark*. The goal of tcpreplay is to provide reliable and repeatable means for testing a variety of network devices such as switches, router, firewalls, network intrusion detection and prevention systems (IDS and IPS).

TCPCopy and tcpreplay are different in the following aspects. First, TCPCopy can be used by high-level applications, while tcpreplay can not. Since tcpreplay is unable to synchronize Seq/Ack numbers or maintain any TCP state, it is unable to establish a valid TCP session with a server. Thus, it is ineffective for high-level applications. However, TCPCopy can capture the online flow, send the captured data to a target server in real-time and establish valid TCP sessions with the target server, so it can be used by high-level applications.

Second, compared with tcpreplay, TCPCopy has less influence on the online server. Tcpreplay often works with packets captured and saved by sniffers/ packet capturing tools, such as *tcpdump*. When the online system is busy, *tcpdump* may influence the online file IO and the packet loss rate may be high. Moreover, as the captured packets should be saved, *tcpdump* seldom captures packets continuously for a few days. On the contrary, TCPCopy does not need many file IO operations, and can copy online packets to a target server in real-time for several days.

Third, TCPCopy keeps more online features than tcpreplay. When replaying with tcpreplay, the source IP address in the packet may be modified and different from online environments. Also it is hard for tcpreplay to keep the packets' online delay feature precisely because of the timing issues, which makes the replaying effect different from online environments. By contrast, TCPCopy captures and sends packets in real-time, so the packets sent to the target server may keep many characteristics of online packets, such as delay and client packet attributes.

Lastly, it is more complex to use tcpreplay than TCPCopy. Compared with tcpreplay, we think TCPCopy is easier to learn and easier to use.

In summary, TCPCopy could be used to test your applications on the TCP layer or above effectively with good performance.

4 How to Use TCPCopy?

4.1 Getting TCPCopy Installed

To use TCPCopy in your work, you first have to install it.

To install TCPCopy:

1) **$** Download *TCPCopy*

Download the latest release (https://github.com/wangbin579/tcpcopy/releases/tag/)

or

Clone the repo: `` `git clone git://github.com/wangbin579/tcpcopy.git` ``.

2) **$** *cd tcpcopy*

2) **$** *sh autogen.sh*

3) **$** *./configure*

4) **$** *make*

5) **#** *make install*

Advanced Options: There are quite a few configuration options for TCPCopy which allow you to control a lot of things. They are as follows.

--enable-debug	compile TCPCopy with debug support (saved in a log file)
--enable-mysqlsgt	run TCPCopy at mysql skip-grant-tables mode
--enable-mysql	run TCPCopy at mysql mode
--enable-offline	run TCPCopy at offline mode
--enable-pcap	run TCPCopy at pcap mode
--enable-nfqueue	run the TCPCopy server (*intercept*) at *nfqueue* mode
--enable-advanced	run TCPCopy with advanced architecture
--enable-dlinject	send packets at the data link layer instead of the network layer
--enable-rlantency	add more lantency control

Recommended use:

1) Recommended use of TCPCopy with traditional architecture

./configure

2) Recommended use of TCPCopy with advanced architecture

./configure --enable-advanced --enable-pcap

3) Recommended use of mysql replay

,/configure --enable-mysqlsgt

It should be noticed that mysql in the test server needs to work in skip-grant-table mode

4) Use of offline replay (TCPCopy also supports offline replay of TCP stream which reads packets from the pcap file)

./configure --enable-offline

.

4.2 Usage guide with traditional architecture

Assume TCPCopy with "./configure" is configured.

After installing TCPCopy, you have to deploy the TCPCopy client (*tcpcopy*) on the online source server and the TCPCopy server (*intercept*) on the target server.

On the target server (root privilege is required):

Using *ip queue*（kernel < 3.5）:

1) *# modprobe ip_queue # if not running*

2) *# iptables -I OUTPUT -p tcp --sport port -j QUEUE # if not set*

3) *# ./intercept*

Or

Using *nfqueue* (kernel >=3.5):

1) *# iptables -I OUTPUT -p tcp --sport port -j NFQUEUE # if not set*

2) *# ./intercept*

On the online source server (root privilege is required):

 # ./tcpcopy -x localServerPort-targetServerIP:targetServerPort

4.3 Usage guide with advanced architecture

Assume TCPCopy with "./configure --enable-advanced --enable-pcap" is configured on the online server and the assistant server.

On the test server (root privilege is required):

Set route commands appropriately to route response packets to the assistant server

On the assistant server (root privilege is required):

./intercept -F <filter> -i <device,>

On the online source server (root privilege is required):

./tcpcopy -x localServerPort-targetServerIP:targetServerPort -s <intercept server,> -i <device,>

Note that the filter format is the same as pcap filter.

For example:

./intercept -i eth0 -F 'tcp and src port 11511' -d

intercept will capture response packets of the TCP based application which listens on port 11511 from device eth0

4.4 Parameters for the TCPCopy Server (*intercept*)

You could use "*intercept -h*" to get the following help information.

-x <passlist,>	passed ip list through firewall Format: ip_addr 1, ip_addr 2 … Only valid with traditional architecture
-p <num>	set the TCP port number to listen on. The default number is 36524.
-l <file>	save log information in <file>
-i <device,>	set the name of the interface to listen on. This is usually a driver name followed by a unit number. For example, eth0 means the first Ethernet interface. Only valid with "./configure --enable-advanced --enable-pcap"
-F <filter>	user filter (same as pcap filter) Only valid with "./configure --enable-advanced --enable-pcap"
-P <file>	save PID in <file>, only used with -d option
-b < ip_addr >	interface to listen on (default: INADDR_ANY, all addresses)
-v	intercept version
-h	print this help and exit

-d run as a daemon

4.5 Parameters for the TCPCopy Client (*tcpcopy*)

On the online server, you could use "*tcpcopy -h*"to get the following help information.

If --disable-dlinject, then

-x <transfer,> use <transfer,> to specify the IPs and ports of the source and target servers. Suppose 'sourceIP' and 'sourcePort' are the IP and port number of the source server you want to copy from, 'targetIP' and 'targetPort' are the IP and port number of the target server you want to send requests to, the format of <transfer,> could be as follows: 'sourceIP:sourcePort-targetIP:targetPort,...'. Most of the time, sourceIP could be omitted and thus <transfer,> could also be: 'sourcePort-targetIP:targetPort,...'. As seen, the IP address and the port number are segmented by ':' (colon), the sourcePort and the targetIP are segmented by '-', and two 'transfer's are segmented by ',' (comma). For example, './tcpcopy -x 80-192.168.0.2:18080' would copy requests from TCP port '80' on current server to the target port '18080' of the target IP '192.168.0.2'.

else

-x <transfer,> use <transfer,> to specify the IPs, ports and MAC addresses of the source and target. The format of <transfer,> could be as follow: 'sourceIP:sourcePort@sourceMac-targetIP:targetPort@targetMac,...'. Most of the time, sourceIP could be omitted and thus <transfer,> could also be: sourcePort@sourceMac-targetIP:targetPort@targetMac,...'. Note that sourceMac is the MAC address of the interface where packets are going out and targetMac is the next hop's MAC address.

endif

-F <filter> user filter (same as pcap filter)
Only valid with "./configure --enable-pcap"

-c < ip_addr > change the client IP to this IP address when sending to the target server. For example, './tcpcopy -x 8080-192.168.0.2:8080 -c 192.168.0.1' would copy requests from port '8080' of current online server to the target port

'8080' of target server '192.168.0.2' and modify the client IP to be '192.168.0.1'.

-C <num> parallel connections between tcpcopy and intercept.
The maximum value allowed is 16(default 3 connections since 0.8.0)

If --enable-pcap, then
-i <device,> The name of the interface to listen on. This is usually a driver name followed by a unit number, for example eth0 for the first Ethernet interface.
else if --enable-offline, then
-i <file> set the pcap file used for tcpcopy to <file>
endif

-o <device,> The name of the interface to send. This is usually a driver name followed by a unit number, for example eth0 for the first Ethernet interface.
Only valid with "./configure --enable-dlinject"

-s <server,> intercept server list
Format:
ip_addr1:port1, ip_addr2:port2,...

-n <num> use <num> to set the replication times when you want to get a copied data stream that is several times as large as the online data. The maximum value allowed is 1023. As multiple copying is based on port number modification, the ports may conflict with each other, in particular in intranet applications where there are few source IPs and most connections are short. Thus, tcpcopy would perform better when less copies are specified. For example,
'./tcpcopy -x 80-192.168.0.2:8080 -n 3' would copy data flows from port '80' on the current server, generate data stream that is three times as large as the source data, and send these requests to the target port '8080' on '192.168.0.2'.

-f <num>	use this parameter to control the port number modification process and reduce port conflictions when multiple tcpcopy instances are running. The value of <num> should be different for different tcpcopy instances. The maximum value allowed is 1023.
-m <num>	set the maximum memory allowed to use for tcpcopy in megabytes, to prevent tcpcopy occupying too much memory and influencing the online system. When the memory exceeds this limit, tcpcopy would quit automatically. The parameter is effective only when the kernel version is 2.6.32 or above. The default value is 512.
-M <num>	MTU value sent to backend (default 1500)
-t <num>	set the session timeout limit. If tcpcopy does not receive response from the target server within the timeout limit, the session would be dropped by tcpcopy. When the response from the target server is slow or the application protocol is context based, the value should be set larger. The default value is 120 seconds
-l <file>	save the log information in <file>
-r <num>	set the percentage of sessions transferred (integer range:1~100)
-p <num>	set the target server listening port. The default value is 36524.
-P <file>	save PID in <file>, only used with -d option
-h	print this help and exit
-v	version
-d	run as a daemon

4.6 An Example (TCPCopy with traditional architecture)

Suppose there are two online servers, 1.2.3.25 and 1.2.3.26. And 1.2.3.161 is the target server. Port 11311 is used as the online source server port and port 11511 is used as the target server port. We use TCPCopy to test if 1.2.3.161 can process 2X requests than an online server can serve.

Here we use TCPCopy to perform the above test task.

1) Download and install TCPCopy

TCPCopy with "./configure" is configured.

2) on the target server (1.2.3.161, kernel 2.6.18)

 # modprobe ip_queue

 # iptables -I OUTPUT -p tcp --sport 11511 -j QUEUE

 # ./intercept

3) on the source online server (1.2.3.25)

 # ./tcpcopy -x 11311-1.2.3.161:11511

4) on the source online server (1.2.3.26)

 # ./tcpcopy -x 11311-1.2.3.161:11511

CPU load and memory usage is as follows:

1.2.3.25:

21158 appuser 15 0 271m 226m 756 S 24.2 0.9 16410:57 asyn_server

9168 root 15 0 18436 12m 380 S 8.9 0.1 40:59.15 tcpcopy

1.2.3.26:

16708 appuser 15 0 268m 225m 756 S 25.8 0.9 17066:19 asyn_server

11662 root 15 0 17048 10m 372 S 9.3 0.0 53:51.49 tcpcopy

1.2.3.1161:

27954 root 15 0 284m 57m 828 S 58.6 1.4 409:18.94 asyn_server

1476 root 15 0 14784 11m 308 S 7.7 0.3 49:36.93 intercept

Access log analysis:

1.2.3.25:

$ wc -l access_1109_09.log

7867867, 2185 reqs/sec

1.2.3.26:

$ wc -l access_1109_09.log

7843259, 2178 reqs/sec

1.2.3.161:

$ wc -l access_1109_09.log

15705229, 4362 reqs/sec

request loss ratio:

(7867867 + 7843259 - 15705229) / (7867867 + 7843259) = 0.0375%

Clearly, the target server can process 2X of requests a source server can serve.

How is the CPU load? Well, the TCPCopy client (*tcpcopy*) on online server 1.2.3.25 used

8.9%, server 1.2.3.26 used 9.3%, while the TCPCopy server (*intercept*) on the target server consumed about 7.7%. We can see that the CPU load is low here, and so is the memory usage.

5 Influential Factors

There are several factors that could influence TCPCopy, which will be introduced in detail in the following sections.

5.1 Capture Interface

TCPCopy utilizes *raw socket input* interface by default to capture packets at the network layer on the online server. The system kernel may lose some packets when the system is busy. Thus, the related system parameters should be set appropriately.

If you configure TCPCopy with "--enable-pcap", then TCPCopy could capture packets at the data link layer and could also filter packets in the kernel.

5.2 Sending Interface

TCPCopy utilizes *raw socket output* interface by default to send packets at the network layer to a target server. The system kernel may encounter problems and not send all the packets successfully. For example, when the packet size is larger than MTU, *raw socket output* interface would refuse to send these large packets. In TCPCopy 0.5 or above versions, with our special processing, large packets are supported.

If you configure TCPCopy with "--enable-dlinject", then TCPCopy could send packets at the data link layer to a target server.

5.3 On the Way to the Target Server

When a packet is sent by the TCPCopy client (*tcpcopy*), it may encounter many challenges before reaching the target server. As the source IP address in the packet is still the end-user's IP address other than the online server's, some security devices may take it for an invalid or forged packet and drop it. In this case, when you use *tcpdump* to capture packets on the target server, no packets from the expected end-users will be captured. To know whether you are under such circumstances, you can choose a target server in the same network segment to do a test. If packets could be sent to the target server successfully in the same network segment but unsuccessfully across network segments, your packets may be dropped halfway.

To solve this problem, we suggest deploying the TCPCopy client (*tcpcopy*) and the TCPCopy server (*intercept*) on servers in the same network segment. There's also another solution with the help of a proxy in the same network segment. The TCPCopy client could send packets to the proxy and then the proxy would send the corresponding requests to the target server in another network segment.

Note that deploying the TCPCopy server on one virtual machine in the same segment may face the above problems.

5.4 OS of the Target Server

The target server may set *rpfilter*, which would check whether the source IP address in the packet is forged. If yes, the packet will be dropped at the network layer.

If the target server could not receive any requests although packets can be captured by *tcpdump* on the target server, you should check if you have any corresponding *rpfilter* settings. If set, you have to remove the related settings to let the packets pass through the network layer.

There are also other reasons that cause TCPCopy not working, such as iptables setting problems in the traditional architecture.

5.5 Applications on the Target Server

It is likely that the application on the target server could not process all the requests in time. On the one hand, bugs in the application may make the request not be responded for a long time. On the other hand, some protocols above TCP layer may only process the first request in the socket buffer and leave the remaining requests in the socket buffer unprocessed.

5.6 *Netlink Socket* Interface[1]

The following problem only occurs in the traditional architecture when IP Queue is used.

Packet loss also occurs when *ip queue* module transfers the response packet to the TCPCopy server (*intercept*) under a high-pressure situation. By using command "cat /proc/net/ip_queue", you can check the state of *ip queue*.

If the value of *queue dropped* increases continually, ip_queue_maxlen should be set larger. For example, the following command modifies the default queue length 1024 to 4096.

echo 4096 > /proc/sys/net/ipv4/ip_queue_maxlen

If the value of *netlink dropped* increases continually, rmem_max and wmem_max should be

[1] intercept uses *ip queue* mode by default for kernel < 3.5

set larger. Here is an example.

sysctl -w net.core.rmem_max=16777216

sysctl -w net.core.wmem_max=16777216

6 Release History

2011.09 version 0.1, TCPCopy released

2011.11 version 0.2, fix some bugs

2011.12 version 0.3, support mysql copy

2012.04 version 0.3.5, add support for multiple copies of the source request

2012.05 version 0.4, fix some bugs

2012.07 version 0.5, support large packets (>MTU)

2012.08 version 0.6, support offline replaying from pcap files to the target server

2012.10 version 0.6.1, support intercept at multi-threading mode

2012.11 version 0.6.3, fix fast retransmitting problem

2012.11 version 0.6.5, support *nfqueue*

2013.03 version 0.7.0, support lvs

2013.06 version 0.8.0, support new configure option with "configure --enable-advanced" for advanced users and optimize intercept

2013.08 version 0.9.0, pcap injection is supported , GPLv2 code has been removed for mysql replay and fixed some bugs

7 Need Help?

Have a bug or a feature request? Please open a new issue.