The author of this program is Salvatore Sanfilippo. He is a hacker from Agrigente, Italy. So all kudos go to him for crafting such an excellent tool. He is presently working on v3, and hopefully will be releasing it soon. There are several tutorials on the web regarding this tool. A couple are by the author himself. However I found them to be confusing, and often difficult to follow. This is no fault of the author as his mother tongue is not English.

The reason I chose to learn this tool is very simple. I was curious as to how the people who were attempting to gain access to our networks were going about it. One of the ways is packet crafting. Crafting packets will allow you to probe firewall rule-sets and find entry points into the targeted system or network.

The sheer versatility of this tool though is what makes it stand out from the crowd. It is not only the king of the hill when it comes to crafting packets. This will show you how to interpret the various conventions of TCP/IP. You will learn how the guts of the stack works by crafting packets. As mentioned below you will need to run a packet sniffer to see your output. This will show you exactly what it is that you are sending out. More importantly though it will show you how various firewalls, open services, and closed ports react to certain stimulus. This side benefit of HPing and packet crafting in general is highly underrated. To understand how the web lives and breathes is integral to your computer studies. It stand to reason that you would want to understand it's basic form of communication ie: the packet. The packet is the foundation upon which C2C (computer to computer) communications are built upon.

The following presentation will show you how to use this tool. It will not however teach you how to hack or to help secure your network. You can do both with HPing. To do both successfully you will need a lot more knowledge in regards to TCP/IP, routers, access control lists, OSI chart, and other areas.

What I hope to accomplish by this brief is to show you just how easy it is to craft packets, and perhaps give you a glimpse into the worldof the black-hat hacker. Not to mention hopefully stimulate your curiousity, and encourage you to further explore the murky world of the hacker. The one constant with hackers of all stripes, whether they be black/white/grey hat is that they have a burning curiosity about computers.

One last note on HPing before we start to look at it. HPing will run on any Linux distro, as well as Net/Free/OpenBSD systems, and lastly it will run on Solaris as well. I highly advise you to run tcpdump at the same time. This will allow you to monitor your crafted packets as well as look at your return packets.

I have included tcpdump snippets to highlight what the outgoing and incoming packets look like on the wire. I believe this to be important as it allows you to visualize the packets.Be aware that HPing "does not" run under Windows. You can however still have tcpdump for Windows. It is called windump and can be found at http://windump.polito.it/

HPing itself can be found at http://www.hping.org/ and lastly tcpdump can be found at http://www.tcpdump.org/

The two packets you see below are just one ip addy sending a Syn packet to another ip addy. To do this using Hping is a very simple task. Just type in "exactly" the below noted command syntax, and voila a syn packet is sent!

hping -S 10.1.1.1

```
19:54:34.236334 10.1.1.1.1321 > 10.1.1.1.0: S 897957123:897957123(0) win 512
0x0000   4500 0028 6510 0000 4006 24bd xxxx xxxx      E..(e...@.$..r.|
0x0010   xxxx xxxx 0529 0000 3585 bd03 67f0 e290      .r...)..5...g...
0x0020   5002 0200 7aac 0000                          P...z...

19:54:36.227147 10.1.1.1.1323 > 10.1.1.1.0: S 576193543:576193543(0) win 512
0x0000   4500 0028 8711 0000 4006 02bc xxxx xxxx      E..(....@....r.|
0x0010   xxxx xxxx 052b 0000 2258 0407 0738 40dd      .r...+.."X...8@.
0x0020   5002 0200 4940 0000                          P...I@..
```

The syn packet is the first step in the TCP/IP hand shake. To open communications between two computers the very first step is to send a syn packet to the computer you wish to communicate with. This would be followed by the syn/ack, in turn followed by the ack. At this point you are ready for the setup of communications, and then the exchange of data.

This is one of the most common scans out there today. The Syn scan is also noisy and easily detectable by properly configured Intrusion Detection Systems such as Snort, Cisco Pix, amongst many others. It is a good scan to run however due to the conventions of TCP/IP. As mentioned above when an open port with a service running upon it receives a syn packet it will respond with a syn/ack confirming it is open and ready for business. Think of port 25 SMTP, port 110 POP3 amongst others that would respond to this type of scan. Assuming that they are not hidden behind firewalls that is. Even if they are though the lack of a response is invaluable information in and of itself. More on that when we look at the rst packet.

Be aware that when you do not specify a destination port on the targeted computer it will default to 0. Also if you do not specify a source port it will use a random ephemeral port and go up numerically from there. More on how to specify both src/dst ports later. On with the basics for now.

The below noted packet is a reset packet. The reset packet is used to reset a connection. As you can see the command syntax is very similar. The only change is in the actual switch itself. Instead of -S it is -R.

The rst packet is often used to perform what is known as inverse mapping. What this means is that rst packets are sent out and the response received is what will tell you if the host exists or not. If you send out a rst scan you would get one of two things. You will

either get no response which will indicate to you that the host is probably alive or an ICMP host unreachable msg. This would indicate that the host does not exist. This is what is known as inverse mapping. Some IDS systems will not log rst packets/scans due to the sheer multitude of them. This is why the inverse scan is popular.

hping -R 10.1.1.1

19:54:57.669980 10.1.1.1.1239 > 10.1.1.2.0: R 1975237774:1975237774(0) win 512
0x0000 4500 0028 890e 0000 4006 00bf xxxx xxxx E..(....@....r.|
0x0010 xxxx xxxx 04d7 0000 75bb bc8e 631c a4e4 .r......u...c...
0x0020 5004 0200 7dbb 0000 P...}..

19:54:58.666747 10.1.1.1.1240 > 10.1.1.2.0: R 225427189:225427189(0) win 512
0x0000 4500 0028 1b76 0000 4006 6e57 xxxx xxxx E..(.v..@.nW.r.|
0x0010 xxxx xxxx 04d8 0000 0d6f bef5 458a 2a1c .r.......o..E.*.
0x0020 5004 0200 7bfa 0000 P...{...

This packet is a fin packet ie: to close a conx already established. Once again the command syntax is very similar to the above two examples. The fin packet is also used to perform the fin scan. This is used against *nix based stacks. When a closed port receives a fin packet it should respond with a rst packet. This will tell you that there are services on the machine, however they are behind a f/w. Once again to my knowledge this is only good against *nix based stacks, and of no use in Windows. Anyone feel free to correct me here.

hping -F 10.1.1.1

19:55:24.992034 10.1.1.1.1502 > 10.1.1.2.0: F 1235235694:1235235694(0) win 512
0x0000 4500 0028 991c 0000 4006 f0b0 xxxx xxxx E..(....@....r.|
0x0010 xxxx xxxx 05de 0000 49a0 336e 3917 cbd5 .r......I.3n9...
0x0020 5001 0200 3507 0000 P...5...

19:55:25.986754 10.1.1.1.1503 > 10.1.1.2.0: F 1775365876:1775365876(0) win 512
0x0000 4500 0028 ccb0 0000 4006 bd1c xxxx xxxx E..(....@....r.|
0x0010 xxxx xxxx 05df 0000 69d1 eef4 00c1 98f3 .r......i.......
0x0020 5001 0200 c486 0000 P.......

The below noted packet is an icmp echo request ie: ping. This packet is useful to determine whether or not a specific host is up or not. The command syntax is a little different for this packet. We will specify after hping that we want a icmp packet by putting in the numerical value 1 followed by the ip addy of the host we are pinging. There are a great many uses for icmp as well as various types of icmp.
This protocol is often used for the ubiquitous ping scan as mentioned above. The obvious reason for this is to see if specific hosts are alive. Nuff said.

hping -1 10.1.1.1

19:55:46.914365 10.1.1.2 > 10.1.1.1: icmp: echo request
0x0000 4500 001c 20cf 0000 4001 690f xxxx xxxx E.....@.i..r.|
0x0010 xxxx xxxx 0800 13e6 e419 0000 .r.....

19:55:47.906748 10.1.1.2 > 10.1.1.1: icmp: echo request
0x0000 4500 001c ee83 0000 4001 9b5a xxxx xxxx E.....@..Z.r.|
0x0010 xxxx xxxx 0800 12e6 e419 0100 .r.....

The packet below is simply a udp packet. To send one is rather easy as well. We will have to tell hping that we want a udp packet by putting in the numerical value 2. The default protocol for hping is tcp. This is why of course we need to tell hping what protocol we wish to send by changing the value.This is of use when probing services which are udp based vice tcp. Such as netbios, nfs, dns, nis amongst others.

hping -2 10.1.1.1

19:56:39.753975 10.1.1.1.2462 > 10.1.1.2.0: udp 0
0x0000 4500 001c c394 0000 4011 c639 xxxx xxxx E.....@..9.r.|
0x0010 xxxx xxxx 099e 0000 0008 053d .r.....=

19:56:40.746747 10.1.1.1.2463 > 10.1.1.2.0: udp 0
0x0000 4500 001c bdc2 0000 4011 cc0b xxxx xxxx E.....@....r.|
0x0010 xxxx xxxx 099f 0000 0008 053c .r.....<

The below noted packet is a syn packet directed at port 21 aka ftp. To send a syn packet at a specific port requires a few more switches. This is where the usage of hping begins to shine. As noted below we are sending a syn (-S) packet to 10.1.1.1 specifically on their ftp port by putting in the (-p) switch. To specify the destination port you put in the -p. To specify the source port on your machine you want the packet to go out on you would use the -s switch followed by a port number just as the destination port example below.

hping -S 10.1.1.1 -p 21

19:57:01.789384 10.1.1.1.1548 > 10.1.1.2.21: S 1371884204:1371884204(0) win 512
0x0000 4500 0028 0661 0000 4006 836c xxxx xxxx E..(.a..@..l.r.|
0x0010 xxxx xxxx 060c 0015 51c5 4aac 669b 5b07 .r.....Q.J.f.[.
0x0020 5002 0200 58aa 0000 P...X...

19:57:02.786747 10.1.1.1.1549 > 10.1.1.2.21: S 1979208427:1979208427(0) win 512
0x0000 4500 0028 d63a 0000 4006 b392 xxxx xxxx E..(.:..@....r.|
0x0010 xxxx xxxx 060d 0015 75f8 52eb 364e 01c6 .r.....u.R.6N..
0x0020 5002 0200 b5c5 0000 P.....

The below noted is a push packet directed at a specific port. In this case http port 80. The "payload" in the push packet should be done up ahead of time in a file that you will specify in the command string. You will as well have to make sure that the packet length is long enough to handle your payload. Hence another switch. I will go over and explain each switch one by one for this type of packet.

-P Tells hping to send a push packet

10.1.1.1 This is the destination ip

-d Allows you +/- the size of the packet itself in this
case we have set it to 80 bytes
-p Specifies the destination port in this case port 80

-E Tells hping where to look for a file which it is to
insert as a payload ie: /home/don/test.sig Quite usefull
obviously for pre-compiled exploits ie: buffer overuns

hping -P 10.1.1.1 -d 80 -p 80 -E /home/don/test.sig

```
19:58:25.721579 10.1.1.1.2426 > 10.1.1.2.80: P 729845249:729845329(80) win 512
0x0000   4500 0078 4a7b 0000 4006 3f02 xxxx xxxx      E..xJ{..@.?..r.|
0x0010   xxxx xxxx 097a 0050 2b80 8e01 5ce6 ac80      .r...z.P+...\...
0x0020   5008 0200 4178 0000 4745 5420 632b 6469      P...Ax..GET.c+di
0x0030   720a 4745 5420 432b 4449 520a 2f2f 6874      r.GET.C+DIR.//ht
0x0040   7470 2031 2e30 0a00 0000 0000 0000 0000      tp.1.0.........
0x0050   0000 0000 0000 0000 0000 0000 0000 0000      ................
0x0060   0000 0000 0000 0000 0000 0000 0000 0000      ................
0x0070   0000 0000 0000 0000                          ........

19:58:26.716801 10.1.1.1.2427 > 10.1.1.2.80: P 823113587:823113667(80) win 512
0x0000   4500 0078 732c 0000 4006 1651 xxxx xxxx      E..xs,..@..Q.r.|
0x0010   xxxx xxxx 097b 0050 310f b773 7ce2 c9a0      .r...{.P1..s|...
0x0020   5008 0200 d559 0000 4745 5420 632b 6469      P....Y..GET.c+di
0x0030   720a 4745 5420 432b 4449 520a 2f2f 6874      r.GET.C+DIR.//ht
0x0040   7470 2031 2e30 0a00 0000 0000 0000 0000      tp.1.0.........
0x0050   0000 0000 0000 0000 0000 0000 0000 0000      ................
0x0060   0000 0000 0000 0000 0000 0000 0000 0000      ................
0x0070   0000 0000 0000 0000                          ........
```

I will now show you how to do what is called Idle Host Scanning. What this means exactly is that we are using one machines ip addy to scan the target computer for open services. To simplify I will use a random ip addy to scan another addy. This I will do by using HPing of course. The one other caveat is that someone's machine needs to be idle.

Bythat I mean not being used by him. This is needed because while I am spoofing his address and sending syn packets to my target I will be sending syn packets as well to his machine to monitor his IP Id numbers. It is through the monitoring of said numbers that we will know if the target machine has open services or not.
When a machine is idle and you send syn packets to it the IP Id numbers will normally go up in a predictable sequence. If the sequence varies it is because the host is now active. By this I mean that the target machine will send to his computer a syn/ack. His machine will respond with an ack packet. This communication between the two will cause the IP Id numbers to change from it's predictable sequence. Thus indicating to us that our spoofed machine has found an open port. All this is done without exposing ourselves to the target machine.

The stacks that are generating predictable IP ID numbers are Win boxes, and older distros of linux such as RedHat 6.1 I beleive. Not sure about the exact distro. It is easy enough to check though. To accomplish said attack we will need to have two sessions of Hping going as well as tcpdump running.

1st session of HPing will contain the below command syntax

hping -S 10.1.1.2 -a 10.1.1.1 -p ++21

-S This again is a syn packet

10.1.1.2 is our target machine

-a The switch used to spoof an ip addy

10.1.1.1 Is the spoofed addy ie: his computer

-p The switch used to specify destination port

++21 Tells hping to syn packet port 21 on up sequentially

The 2nd session of HPing will contain the below noted command syntax

hping -1 10.1.1.1

By sending icmp packets to his machine I will get back the info I need to execute this. I will get back ttl's and more importantly of course the IP Id numbers. I will keep pinging his box all the while I am sending spoofed syn packets to the target machine in the hope they respond. This will result in his machine changing it's IP Id numbers from it predictable sequence. Thus indicating thatit has found an open port. Be aware though that this will only work with a middle man with whom you can monitor it's IP Id numbers. If you have a machine which is running no services, and is firewalled this will not work.

Seen as any packets icmp/syn or otherwise will simply be dropped. Your best bet would be to ask someone you know who has a broadband account if they would be willing to let you experiment with their machine. Either way here is a url that does an excellent job of explaining the IP ID attack. There are many more out there just google for them. http://www.bursztein.net/secu/temoinus.html

I have included on the next page some examples of HPing strings and the feedback as well as the tcpdump logs. Feel free to experiment with the below noted. Not just that mess around with fragmented packets, setting your X and Y flags and the like. You will only learn by playing around. Ideally get a friend that you can bounce this stuff to, and later go over you results with him/her. That is it for now folks. I hope I was able to edumacate ya some!

These examples are of IP addy's that were bouncing off of my f/w. So I decided to return the favour and take a looksie for fun. As you can see port 25 SMTP ack'ed back. In case your wondering what all that gobbledeegook is at the end of the syn/ack packet let me explain.

S denotes a syn packet

973256460:973256460 is the sequence number of the packet

ack 1440279771 is the ack sequence number

win 16616 is the amount of buffered space the machine has to receive info

<mss 1460> means the maximum segment size is 1460 this relates to the mtu

mtu means the maximum transmission unit <which is a maximum of 1500>

HPING 24.114.15.186 (eth0 24.114.15.186): S set, 40 headers + 0 data bytes

len=46 ip=24.114.15.186 sport=21 flags=RA seq=0 ttl=122 id=20284 win=0
rtt=48.8 ms
len=46 ip=24.114.15.186 sport=22 flags=RA seq=1 ttl=122 id=20367 win=0
rtt=34.3 ms
len=46 ip=24.114.15.186 sport=23 flags=RA seq=2 ttl=122 id=20505 win=0
rtt=24.9 ms
len=46 ip=24.114.15.186 sport=24 flags=RA seq=3 ttl=122 id=20672 win=0
rtt=25.8 ms
len=46 ip=24.114.15.186 sport=25 flags=SA DF seq=4 ttl=122 id=20767
win=16616 rtt=45.8 ms
len=46 ip=24.114.15.186 sport=26 flags=RA seq=5 ttl=122 id=20879 win=0
rtt=35.8 ms
len=46 ip=24.114.15.186 sport=27 flags=RA seq=6 ttl=122 id=20937 win=0

20:37:18.131546 10.1.1.1.1238 > 24.114.15.186.25: S 1440279770:1440279770(0) win 512
0x0000 4500 0028 1061 0000 4006 7755 xxxx xxxx E..(.a..@.wU.r.|
0x0010 1872 0fba 04d6 0019 55d8 ecda 748e 5412 .r.....U...t.T.
0x0020 5002 0200 aa85 0000 P.....

20:37:18.177151 24.114.15.186.25 > 10.1.1.1.1238: S 973256460:973256460(0) ack 1440279771 win 16616 <mss 1460> (DF)
0x0000 4500 002c 511f 4000 7a06 bc92 1872 0fba E..,Q.@.z....r..
0x0010 xxxx xxxx 0019 04d6 3a02 b70c 55d8 ecdb .r.|....:...U...
0x0020 6012 40e8 2b62 0000 0204 05b4 0000 `.@.+b.....

The same person bouncing off of my f/w. Port 25 SMTP still acking

monkey:/home/don # hping -S 24.114.15.186 -s 1234 -p ++21 -f

HPING 24.114.15.186 (eth0 24.114.15.186): S set, 40 headers + 0 data bytes

len=46 ip=24.114.15.186 sport=21 flags=RA seq=0 ttl=122 id=100 win=0 rtt=52.9 ms
len=46 ip=24.114.15.186 sport=22 flags=RA seq=1 ttl=122 id=101 win=0 rtt=27.0 ms
len=46 ip=24.114.15.186 sport=23 flags=RA seq=2 ttl=122 id=102 win=0 rtt=31.4 ms
len=46 ip=24.114.15.186 sport=24 flags=RA seq=3 ttl=122 id=103 win=0 rtt=47.4 ms
len=46 ip=24.114.15.186 sport=25 flags=SA DF seq=4 ttl=122 id=104 win=16616 rtt=28.1 ms
len=46 ip=24.114.15.186 sport=26 flags=RA seq=5 ttl=122 id=105 win=0 rtt=41.1 ms
len=46 ip=24.114.15.186 sport=27 flags=RA seq=6 ttl=122 id=106 win=0 rtt=48.8 ms

Another person this time with port 53 DNS and HTTP port 80 acking back to me

monkey:/home/don # hping -S 24.114.9.226 -s 1234 -p ++21

HPING 24.114.9.226 (eth0 24.114.9.226): S set, 40 headers + 0 data bytes

len=46 ip=24.114.9.226 sport=21 flags=RA seq=0 ttl=118 id=19063 win=0 rtt=227.5 ms
len=46 ip=24.114.9.226 sport=22 flags=RA seq=1 ttl=118 id=19092 win=0 rtt=211.6 ms
len=46 ip=24.114.9.226 sport=23 flags=RA seq=2 ttl=118 id=19132 win=0 rtt=264.6 ms

len=46 ip=24.114.9.226 sport=24 flags=RA seq=3 ttl=118 id=19180 win=0
rtt=419.1 ms
len=46 ip=24.114.9.226 sport=25 flags=RA seq=4 ttl=118 id=19202 win=0
rtt=925.0 ms
len=46 ip=24.114.9.226 sport=53 flags=SA DF seq=32 ttl=118 id=20344
win=16616 rtt=158.2 ms
len=46 ip=24.114.9.226 sport=80 flags=SA DF seq=59 ttl=118 id=21413
win=16616 rtt=151.7 ms

Another person this time with more services open as you can see. Just because services are open does not mean your in like Flynn! Rather you will now have to case the joint. Grab some banners dude!

monkey:/home/don # hping -S 164.77.216.98 -s 3133 -p ++21

HPING 164.77.216.98 (eth0 164.77.216.98): S set, 40 headers + 0 data bytes

len=46 ip=164.77.216.98 sport=21 flags=SA DF seq=0 ttl=49 id=0 win=5840
rtt=339.7 ms
len=46 ip=164.77.216.98 sport=22 flags=SA DF seq=1 ttl=49 id=0 win=5840
rtt=2319.8 ms
len=46 ip=164.77.216.98 sport=25 flags=SA DF seq=4 ttl=49 id=0 win=5840
rtt=516.0 ms
len=46 ip=164.77.216.98 sport=53 flags=SA DF seq=32 ttl=49 id=0 win=5840
len=46 ip=164.77.216.98 sport=80 flags=SA DF seq=59 ttl=49 id=0 win=5840

11:18:34.312598 164.77.216.98.21 > 10.1.1.1.3133: S [tcp sum ok]
3829392622:3829392622(0) ack 1327465401 win 5840 <mss 1460> (DF) (ttl 4
9, id 0, len 44)
0x0000 4500 002c 0000 4000 3106 022e a44d d862 E..,..@.1....M.b
0x0010 xxxx xxxx 0015 0c3d e43f e4ee 4f1f 83b9 .r.|...=.?..O...
0x0020 6012 16d0 914e 0000 0204 05b4 0000 `....N.....

11:19:34.399961 164.77.216.98.80 > 10.1.1.1.3192: S [tcp sum ok]
3889760816:3889760816(0) ack 974078565 win 5840 <mss 1460> (DF) (ttl 49
, id 0, len 44)
0x0000 4500 002c 0000 4000 3106 022e a44d d862 E..,..@.1....M.b
0x0010 xxxx xxxx 0050 0c78 e7d9 0a30 3a0f 4265 .r.|.P.x...0:.Be
0x0020 6012 16d0 be61 0000 0204 05b4 0000 `....a.....

I did not include the tcpdump for all ports who syn/acked
As seen in the above 5 ports which answered

APPENDIX I

```
usage: hping host [options]
  -h  --help      show this help
  -v  --version   show version
  -c  --count     packet count
  -i  --interval  wait (uX for X microseconds, for example -i u1000)
      --fast      alias for -i u10000 (10 packets for second)
  -n  --numeric   numeric output
  -q  --quiet     quiet
  -I  --interface interface name (otherwise default routing interface)
  -V  --verbose   verbose mode
  -D  --debug     debugging info
  -z  --bind      bind ctrl+z to ttl           (default to dst port)
  -Z  --unbind    unbind ctrl+z
Mode
  default mode     TCP
  -0  --rawip      RAW IP mode
  -1  --icmp       ICMP mode
  -2  --udp        UDP mode
  -9  --listen     listen mode
IP
  -a  --spoof      spoof source address
  -t  --ttl        ttl (default 64)
  -N  --id         id (default random)
  -W  --winid      use win* id byte ordering
  -r  --rel        relativize id field          (to estimate host traffic)
  -f  --frag       split packets in more frag.  (may pass weak acl)
  -x  --morefrag   set more fragments flag
  -y  --dontfrag   set dont fragment flag
  -g  --fragoff    set the fragment offset
  -m  --mtu        set virtual mtu, implies --frag if packet size > mtu
  -o  --tos        type of service (default 0x00), try --tos help
  -G  --rroute     includes RECORD_ROUTE option and display the route buffer
  -H  --ipproto    set the IP protocol field, only in RAW IP mode
ICMP
  -C  --icmptype   icmp type (default echo request)
  -K  --icmpcode   icmp code (default 0)
      --icmp-ts    Alias for --icmp --icmptype 13 (ICMP timestamp)
      --icmp-addr  Alias for --icmp --icmptype 17 (ICMP address subnet mask)
      --icmp-help  display help for others icmp options
UDP/TCP
  -s  --baseport   base source port             (default random)
  -p  --destport   [+][+]<port> destination port(default 0) ctrl+z inc/dec
 -k  --keep       keep still source port
```

```
  -w  --win        winsize (default 64)
  -O  --tcpoff     set fake tcp data offset     (instead of tcphdrlen / 4)
  -Q  --seqnum     shows only tcp sequence number
  -b  --badcksum   (try to) send packets with a bad IP checksum
                   many systems will fix the IP checksum sending the packet
                   so you'll get bad UDP/TCP checksum instead.
  -M  --setseq     set TCP sequence number
  -L  --setack     set TCP ack
  -F  --fin        set FIN flag
  -S  --syn        set SYN flag
  -R  --rst        set RST flag
  -P  --push       set PUSH flag
  -A  --ack        set ACK flag
  -U  --urg        set URG flag
  -X  --xmas       set X unused flag (0x40)
  -Y  --ymas       set Y unused flag (0x80)
  --tcpexitcode    use last tcp->th_flags as exit code
  --tcp-timestamp  enable the TCP timestamp option to guess the HZ/uptime
Common
  -d  --data       data size                  (default is 0)
  -E  --file       data from file
  -e  --sign       add 'signature'
  -j  --dump       dump packets in hex
  -J  --print      dump printable characters
  -B  --safe       enable 'safe' protocol
  -u  --end        tell you when --file reached EOF and prevent rewind
  -T  --traceroute traceroute mode             (implies --bind and --ttl 1)
  --tr-stop        Exit when receive the first not ICMP in traceroute mode
  --tr-keep-ttl    Keep the source TTL fixed, useful to monitor just one hop
  --tr-no-rtt      Don't calculate/show RTT information in traceroute mode
```