socat – Handling all Kinds of Sockets

Gerhard Rieger

Linuxwochen

Vienna, 06/01/2007
Motivation: netcat

- “Swiss army knife” of shell/socket tools
- Hobbit 1995?
- public domain?
- makes a socket accessible via stdio
- TCP, UDP
- connect (client mode), listen (server mode)
- run program instead of stdio
- e.g.: `nc -u -l -p 8000 -e /bin/cat`
- portscan (TCP connect)
- telnet controls
netcat - Limitations

- “left“ side: stdio or program
- “right“ side: TCP, UDP – connect, listen
- one-shot only (terminates after socket close)
- direct derivatives:
  - cryptcat blowfish encryption
  - aes-netcat AES encryption
- netcat rewrites:
  - netcat6 IPv6
  - GNU netcat “tunnel“ mode (port forwarder)
  - sbd AES encryption
  - ncat ssl, proxy, socks, IPv6
  - connect socks, proxy client for ssh
Ideas for Extension

- support more socket types (datagram, raw IP)
- support other stream types
  isn't a serial line like a connected socket?
- multiple connections (“server“ or “daemon“ mode)
- symmetric concept: “left“ and “right“ sides both provide same feature set, be fully configurable
- each “address“ is one command line parameter
- pass all low level features to application
  * socket keepalives
  * on error provide information from errno etc.
Implementation: socat

- new from scratch, C language
- GPL2
- takes two “addresses“:
  socat [OPTIONS] address1 address2
- opens both, transfers data forth and back using UNIX filedescriptors
- can handle multiple connections (fork)
- options for system call tracing (debugging, learning)
- traffic dumping in text and/or hex
- dozens of address types
- hundreds of options to addresses
socat Basic Address Types

stdio:
  stdio

TCP client, server:
  tcp4:localhost:1080
  tcp4-listen:8080  # short: tcp-l:...

UDP client, server:
  udp4:host:2049   # -T inactivity timeout
  udp4-l:500

run program in subprocess:
  exec:/bin/ls,pty

socat addresses consist of a keyword, required parameters, and address options:
  keyword:param1:param2,option1,option2,...
Examples 1: netcat Replacement

- TCP client:
  nc 1.2.3.4 25
  socat - tcp:1.2.3.4:25

- UDP client with source port:
  nc -u -p 500 1.2.3.4 500
  socat - udp:1.2.3.4:500,sp=500

- TCP server:
  nc -l -p 8080
  socat - tcp-l:8080,reuseaddr

- TCP server with direct script:
  nc -l -p 7000 -e /bin/cat
  socat tcp-l:7000,reuseaddr exec:"/bin/cat",nofork
existing file descriptors:
  fd:3
open files, devices, named pipes:
  open:hello.txt
  open:/dev/tty
  create:newfile
readline ('bash' line editor):
  readline,history=
    $HOME/.http_history
run program in subshell:
  system:'while read ...
          ...; do ...; done',pipes
proxy connect client:
  proxy:proxy.local:\
    www.remote.com:443
socks4, socks4a client:
  socks:socks.local:\
    www.remote:80
OpenSSL client, server:
  ssl:www.local:443,\
    verify=0
  ssl-l:443,\
    cert=./server.pem
IP v4 and v6:

tcp6:www.harakiri.jp:80
udp:[::1]:123 # autodetects IP vers.
socks4:socks.local:www:80 # '4' is not IPv4!
ssl-l:443,pf=ip6,cert=...

UNIX domain stream client, server
unix-connect:/tmp/.X11-unix/X0
unix-listen:$HOME/dev/socket1,fork

UNIX domain datagram sender, receiver
unix-sendto:/dev/log
unix-recv:/dev/log
unix-recvfrom:$HOME/dev/askmewhat,fork

Abstract UNIX sockets: all above types, e.g.:
abstract-connect:/tmp/dbus-aL7CFhBj5I
socat Address Types 4

“generic open” for file etc, or UNIX socket:
  gopen:data.csv
  gopen:/tmp/X11-unix/X0 /dev/log

UDP sender, receiver:
  udp-sendto:host:123
  udp4-recv:514
  udp6-recvfrom:123,fork
  udp-datagram:host:port

similar for raw IP protocols:
  ip4-sendto:host:53

creates unnamed and named pipes:
  pipe
  pipe:./named.pipe

creates ptys:
  pty,link=$HOME/dev/pty0
Common Uses

- socat: wide range of possibilities by selecting appropriate address types
- specialized tools exist for mainstream purposes
- e.g. netcat, rinetd, rlwrap, socks/proxy clients, stunnel, ser2net
- remote tty, e.g. in Heise c't projects:
  - creates PTY, holds its master side and exchanges data via TCP client
- redirector for mysql client to remote server:
  - UNIX socket listener with TCP client
- access serial device of VMWare guest OS:
  - stdio with UNIX socket client
- external socksifier for Tor:
  - TCP listener with socks4a client
Address Variants

unidirectional mode (-u: left to right, -U: reverse):
```
socat -u stdin stdout
```

combine two addresses to one dual address:
```
stdout%stdin   (socat V1: stdin!!stdout)
```

fork mode with most listening/receiving sockets:
```
tcp4-l:80,fork
udp6-recvfrom:123,fork
```

retry: don't exit on errors, but loop:
```
tcp4-l:8080,retry=10,intervall=7.5
```

some clients with fork or retry:
```
tcp:www.domain.com:80,fork,forever,intervall=60
```

ignoreeof: EOF does not trigger shutdown (tail -f):
```
open:/var/log/messages.log,ignoreeof
```
to each address, many options can be applied. “option groups“ determine if an option can be used with an address
- FD (FD type may be unknown) e.g. locks, uid
- open flags (with open() call)
- named (file system entry related), ext2/ext3/reiserfs attrs
- process options (setuid, chroot)
- readline (history file), termios
- application level (EOL conv, readbytes)
- socket, IP, TCP, DNS resolver options
- socks, HTTP connect parameters (socksuser)
- listen, range, child, fork, retry
- OpenSSL
Address Options 2

- option examples:
  - `perm=700`
  - `bind=192.168.0.1:54321`
  - `proxy-auth=hugo:s3cr3t`
  - `intervall=1.5`

- alias names vs. canonical names:
  - `debug` vs. `so-debug`
  - `async` vs. `o-async`
  - `maxseg` vs. `tcp-maxseg`

- canonical namespace related to C language:
  - C defines: `O_ASYNC`, `ECHO`, `bind()`
  - socat options: `o-async`, `echo`, `bind`

- address and option keywords are case insensitive
Options for OpenSSL Client

- The option groups of an address type determine which options may be used.
- OpenSSL client addresses have these groups:
  - OPENSSL
  - TCP
  - IPAPP
  - IP
  - RETRY
  - SOCKET
  - FD
OpenSSL Options

cipher=[3DES|MD5|...]
method=[SSLv3]
verify=[0|1]  # default: 1
cafile=<filename>  # trusted public certs
cert=<filename>  # cert and private key
egd=<filename>  # socket for entropy
pseudo  # only pseudo random
TCP Options

all OpenSSL options
and:

- `mss=1400`  # maximum segment size
- `nodelay`   # disable Nagle algorithm
- `syncnt`    # max. number of SYN retransmits
- `cork`      # don't send short packets
- `defer-accept`  # accept only when data arrived
- `sack-disable` # OpenBSD
- `noopt`    # FreeBSD
- ...

all OpenSSL and TCP options and:

```plaintext
sourceport=<port>  # client: bind to this port
                   # server: compare with peer
lowport           # client: bind to pseudo random
                   # < 1024, requires root
                   # server: check peer port
```
all OpenSSL and TCP options, all “IP-application“ (port) options and:

```
ttl=...        # time to live
tos=...       # type of service
mtudiscover=[0|1|2]    # see/usr/include/linux/in.h
ipoptions=<data>  # source routing...
...```


General Socket Options

all OpenSSL and TCP options,
all "IP-application" (port) and IP options,
and those that apply to all socket families:

bind=<address>  # bind to port, address
keepalive
recvbuf=<size>
reuseaddr  # recommended?
type=<socktype>  # for socket() call
pf=<protocol family>  # ip4, ip6
sndbuf=131028
connect-timeout=<seconds>
all OpenSSL and TCP options,
all “IP-application“ (port), IP, and socket options,
and those that apply to all file descriptors:

- **nonblock**
  # for connect()

- **shutdown-none**
  # do not shutdown() - for shared sockets

- **cool-write**
  # write failure does not print error, keeps log file clean

many more FD options exist that are not really useful on sockets (user, group, mode, locking, ...)

Example: SSL Tunnel

- Two socat processes communicating via TCP:
  
  **server:**
  
  ```
  socat tcp-listen:8888,reuseaddr,fork
  <some-address1>
  ```

  **client:**
  
  ```
  socat <some-address2>
  tcp-connect:hostname:8888
  ```

- Replace TCP addresses with OpenSSL addresses:
  
  **server:**
  
  ```
  socat ssl-listen:8888,reuseaddr,fork,
  cert=server.pem,cafile=client.crt
  <some-address1>
  ```

  **client:**
  
  ```
  socat <some-address2>
  ssl-connect:hostname:8888,
  cert=client.pem,cacert=client.crt
  ```
Logging, Tracing

socat options:

- `-d` # more debug output (up to 4 times)
- `-ly` # log to syslog
- `-lf <filename>` # log to file
- `-lm` # initial logging goes to stderr, then to syslog
- `-lu` # microsecond timestamps
- `-lp <progname>` # name used in log messages
- `-v` # verbose traffic (text)
- `-x` # verbose traffic (hex)
termios Options

- with tty, with explicit pty, or on exec and system with pty
  - raw  # transparent mode
  - echo=0  # don't echo input
  - b115200  # baud rate
  - icanon  # line buffer, special chars
  - min=1  # pass each char immediately
  - brkint  # ^C triggers SIGINT
  - see „man termios“
  - ctty  # make it controlling terminal
  - setpgid  # make it process group leader
  - setsid  # make a new session
Example: Remote TTY

- **client side:**
  
  ```bash
  socat \\
  pty,link=$HOME/dev/pty0,raw,echo=0,waitslave \\
  tcp:server:54321
  ```

  **PTY server:**
  
  ```bash
  socat tcp-l:54321,fork \\
  /dev/modem,raw,echo=0,waitlock=/var/lock/modem.lock
  ```

  can be changed to use SSL

- **using ssh:**
  
  ```bash
  socat pty,link=$HOME/dev/pty0,raw,echo=0,waitslave \\
  exec:'ssh -T -l user server \\
  "socat - /dev/modem,raw,echo=0"'
  ```
Example: Passing Friendly Firewall

want to reach server in your company's intranet after work?

before leaving, start „double client“:

```
socat
  ssl:priv-host:443,fork,forever,\
    intervall=30,cert=...,cafile=...,verify \
  tcp:protected-server:80
```

at home, start double server:

```
socat
  tcp-l:80,fork \
  tcp-l:443,reuseaddr,forever,cert=...,cafile=...
```

with browser connect to double server

is „loud“ (many FW log entries?)
Security Options 1:
Server Sockets

bind to specific address: bind=10.1.2.3
bind to specific interface: bindtodevice=eth0
sourceport restriction of client sockets:
sp=53
or lowport
client range:
range=10.0.0.0/8
tcpwrappers, custom configuration
tcpwrap=mydns
tcpwrap-etc=$HOME/etc
Security Options 2: Process Options

- all these options require starting with root or equivalent
  - chroot: chroot=/var/chroot/jail
  - just change GID: setgid=<group>
  - just change UID: setuid=<user>
  - impersonate different user (UID, GIDs):
    su=<user>
  - impersonate different user, effective after chroot:
    su-d=<user>
Example: Secured Server Script

• provide a simple service via TCP (or SSL ...), run program with reduced privilege:

```bash
socat \
tcp-l:7,fork,tcpwrap="echol" \
exec:/bin/cat,chroot=/var/sandbox,su-d=nobody,\nttys,setsid,raw,echo=0
```
New Features in 1.6.0 (March 2007)

- New address type UDP-datagram:<host>:<port> allows “symmetric” datagram modes; previous modes were client or server types.
- Also with raw IP protocol
- New option ip-add-membership for multicast support
- New address type TUN for creation of TUN/TAP devices
- Support for abstract UNIX domain sockets (not in file system; name starts with '\0')
- New option end-close allows to keep socket connections open
- Option range now supports form range=address:mask with IPv4 (old: range=address/bits)
- Address OPENSSL-LISTEN now requires client certificate per default (or use option verify=0)
- Lock support extended to distinguish read and write locks.
Examples: Symmetric Broadcast/Multicast Datagrams

- datagram addresses: take IP-address, port; send to this address and receive only from this address (except with range option)

- peers on local network communicate symmetrically (vs. client/server)

- broadcast version:
  ```
  socat - UDP4-DATAGRAM:255.255.255.255:9999,\ 
  bind=:9999,range=192.168.0.0/24
  ```

- multicast version:
  ```
  socat - UDP4-DATAGRAM:224.255.0.1:6666,bind=:6666,\ 
  ip-add-membership=224.255.0.1:eth0,\ 
  range=192.168.0.0/24
  ```
TUN/TAP interfaces

- TUN/TAP: Linux “logical” or “virtual” network interfaces, where a process simulates the wire.

- socat can create such a device, emulate the wire, and transfer the data. Build a simple VPN between two hosts:
  - server:
    ```
    socat tcp-l:50500 TUN:192.168.10.2/24,iff-up=1
    ```
  - client:
    ```
    socat tcp:50500 TUN:192.168.10.3/24,iff-up=1
    ```
- use SSL instead of TCP...
filan File Analyzer

• currently part of socat distribution, will be separated some day

• analyzes its file descriptors

• was developed for debugging socat but might be useful for other purposes too

• can show surprising weaknesses of parent process, e.g. xterm, gdb, socat
filan: gdb Crying for Heisenbugs

• normal process:

```bash
$ filan -s
tty /dev/pts/0
tty /dev/pts/0
tty /dev/pts/0
```

• run in gdb:

```bash
$ gdb filan
(gdb) r -s
Starting program: /usr/local/bin/filan -s
tty /dev/pts/0
tty /dev/pts/0
tty /dev/pts/0
tty /dev/pts/0
pipe
pipe
file /usr/local/bin/filan
```
<table>
<thead>
<tr>
<th>FD</th>
<th>type</th>
<th>device</th>
<th>inode</th>
<th>mode</th>
<th>links</th>
<th>uid</th>
<th>gid</th>
<th>rdev</th>
<th>size</th>
<th>blksize</th>
<th>blocks</th>
<th>atime</th>
<th>mtime</th>
<th>cloexec</th>
<th>flags</th>
<th>sigown</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>socket</td>
<td></td>
<td>0,4</td>
<td>106115</td>
<td>0140777</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>0,0</td>
<td>1024</td>
<td>0</td>
<td>Thu Jan 1 01:00:00 1970</td>
<td>Thu Jan 1 01:00:00 1970</td>
<td>0</td>
<td>DEBUG=0</td>
<td>REUSEADDR=1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TYPE=1</td>
<td>ERROR=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DONTROUTE=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BROADCAST=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SNDBUF=50616</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RCVBUF=87408</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KEEPALIVE=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OOBINLINE=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NO_CHECK=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PRIORITY=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LINGER={0,0}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PASSCRED=0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PEERCRED={0,-1}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RCVLOWAT=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SNDLOWAT=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RCVTIMEO={0,0}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SNDTIMEO={0,0}</td>
<td></td>
</tr>
</tbody>
</table>

**AF=2 127.0.0.1:7777 <-- AF=2 127.0.0.2:32769**

**TCP_NODELAY=0**

**TCP_MAXSEG=16384**

**TCP_KEEPIDLE=7200**

**TCP_KEEPCNT=9**

**TCP_SYNCNT=5**

**TCP_LINGER2=60**

**TCP_ACCEPT=0**

**TCP_WINDOW_CLAMP=49172**

**TCP_INFO={00000001 00220700 00031ce0 00000000 00004000 00000218 00000000 00000000 00000000 00000000 00000000 00000008 00000000 0147e370 00000008 00004034 00007fff 00000fa0 000007d0 7ffffffff 00000002 00004000 00000003 00000000 000007fff 00000000}**

**TCP_QUICKACK=1**
procan Process Analyzer

prints infos about actual process:

process id = 11289
process parent id = 6179
controlling terminal: "/dev/tty"
process group id = 11289
process session id = 6179
process group id if fg process / stdin = 11289
process group id if fg process / stdout = 11289
process group id if fg process / stderr = 11289
process has a controlling terminal
user id = 1000
effective user id = 1000
group id = 1000
effective group id = 1000
Platforms, Porting

- C language, source under GPL2
- development platform: Linux
- test.sh for quick testing of many features
- mainstream Linux's should pass all tests
- passes most tests on common UNIX systems like *BSD, AIX, HP-UX, Solaris
- compiles on Cygwin
- packages available for most Linux dists, for *BSD; Solaris, AIX; on many security Live CDs
nest features (build protocol stack), e.g. use gzip over SSL over proxy-connect over TCP/IPv6:

```
soctat - 'gzip|ssl,cafile=...|proxy:server:9000|tcp6:proxy:8080'
```

this would make lots of new address types interesting:

- socks5
- send or receive data per HTTP, SMTP, ...
- more encryption types, GNU TLS
- telnet controls, recode
- hping mode
- user space IP stack
Future has just begun: socat version 2

- Problem: using SSL over HTTP Proxy – each browser is able to do this!
  workaround:
  ```
  socat TCP-L:50443,reuseaddr,fork PROXY:<proxy>:<target>:443
  socat STDIO SSL:localhost:50443
  ```

- would be nice to combine socat calls

- -> socat version 2:
  ```
  socat STDIO 'SSL|PROXY:<target>:443|TCP:<proxy>:8080'
  ```

- This is implemented in form of multiple socat engines:
  ```
  STDIO <socat> SSL pipes <socat> PROXY pipes <socat> TCP
  ```

- Each engine runs in its own POSIX thread
Address Chains

- socat now accepts two address chains
- Each address chain consists of zero or more inter addresses and one endpoint address (sub addresses)
- Sub addresses have the same syntax as socat V1 addresses: `keyword:param1:param2,option1,option2,...`
- Endpoint addresses: link socat with the outside world using UNIX file descriptor(s); were already available in socat V1: `TCP, UDP, FILE, PTY, EXEC, ...`
- Inter addresses: they manipulate (bidirectional) data streams: `SSL, PROXY`
- For compatibility and simplicity: old combinations of inter and endpoint addresses are still available as endpoint addresses
Building Address Chains

- `socat stdio \n  'socks:targethost.customer.com:22|\n  openssl,cafile=customer.cert|\n  proxy-connect:sockd.customer.com:443|\n  tcp:proxy.domain.com:8080'

- This means – interpreting from back to forth:
  - connect to proxy.domain.com:8080 with TCP
  - send an HTTP CONNECT request for sockd.customer.com:443 to the proxy
  - drive SSL over the proxy to sockd
  - send a socks-Request through the SSL tunnel for targethost.customer.com:22
  - at last make this channel available via stdio
Nested Scripts

- Alternatively to long chains, scripts can be defined and invoked:

  toproxy.sh: exec socat - tcp:proxy.domain.com:8080
  tohome.sh: exec socat - "proxy-connect:sockd.customer.com:443\exec:toproxy.sh"
  sslhome.sh: exec socat - "openssl,cafile=customer.cert\exec:tohome.sh"

- Then run the 'last' script:

  socat - socks:targethost.customer.com:22

- Obviously, the (last) script need not be socat based
Handling of Address Chains

Compare a single address chain with a UNIX file: its textual specification correlates to a filename; it can be opened in different modes. After opening, the UNIX file descriptor of the file correlates to a handle of the address chain. We have some abstract functions:

- handle = open(string, mode)
- read(handle, ...)
- write(handle, ...)
- shutdown(handle)
- close(handle)

socat basically uses these operations on the address chains
Existing Inter Addresses (should become more soon...)

- Some inter address functionality already existed in socat V1 in implicit combination with endpoint addresses; they are now available as inter addresses:
  
  ```
  socks4
  socks4a
  proxy-connect
  ```

- New inter addresses with socat V2:
  
  ```
  nop     // does “nothing” - just transfers data
  test    // bidirectional: appends '>' or '<' to each transferred block
  testuni // unidirectional: appends '>' to each transferred block
  testrev // unidir. reverse: appends '<' to each transferred block
  socks5-client // or just socks5
  ```
Overloading and Effective Data

- for each sub address type, its supported contexts are defined within socat (b...bidirectional):
  
tcp-connect: type=endpoint params=2 left={r,w,b} right=n/a
  nop: type=inter params=0 left={r} right={w}
  type=inter params=0 left={w} right={r}
  type=inter params=0 left={b} right={b}

proxy-connect: type=endpoint params=3 left={r,w,b} right=n/a
  type=inter params=2 left={r,w,b} right={b}

- Old “protocol” addresses vs. new inter addresses
  V1: proxy:proxyserver:targetserver:targetport
  V2: proxy:targetserver:targetport|tcp:proxyserver:8080

- new proxy address assembles and sends proxy CONNECT request, has nothing to do with proxy server address:

  CONNECT targetserver:targetport HTTP/1.0
Reverse Inter Addresses

- Consider the following socat V1 invocation:

  ```bash
  socat openssl-server:443,fork exec:myscript
  ```

  Without resorting to V1 style openssl-server, this would be expressed in V2:

  ```bash
  socat 'openssl-server|tcp-listen:443,fork' exec:myscript
  ```

  It might for some reason be better to have the openssl-server in the right chain.

- Reverting addresses changes their direction of operation:

  ```bash
  socat tcp-listen:443,fork '^openssl-server|exec:myscript'
  ```

- Note: reverting a bidirectional gzip would be equivalent to a bidirectional gunzip, but reverting openssl-server is NOT equivalent to openssl-client
Unidirectional Chains

- socat V2 still provides options -u, -U for unidirectional transfer:
  
  ```
  socat -u 'stdin' 'gzip|stdout'
  socat -U 'gzip|stdout' 'stdin'
  ```

- chain elements are adapted to their context:
  
  ```
  socat -u 'stdio' \ 'openssl-client|tcp:10.11.12.13:443'
  ```

  stdio is only read from

  openssl-client is only written to, but it communicates bidirectionally with its right neighbor
Dual Type Inter Addresses

- socat V1 allowed different sub addresses for input and output ("dual"):
  
socat stdio exec:myscript
socat 0!!1 exec:myscript

- socat V2 extends this idea to inter addresses. Consider a (not yet implemented) gzip inter address that compresses left to right and decompresses right to left:
  
socat stdio 'gzip|...'

If we had a unidirectional gunzip we could type (making gzip unidirectional):

  socat stdio 'gzip%gunzip|...'

First part of dual address from left to right, second part from right to left

- Combined with reverse feature:
  
socat 'gzip%^gzip|stdio' '...'

Again: left-to-right%right-to-left
Dual Addresses: Incompatible Change from socat V1

• in socat V1, the first part of a dual address was for reading and the second for writing (read / write); this was considered to be intuitive because:
  stdio was equivalent to 0!!1
  and because the pipe(2) call returns its FDs in the filedes array in the order read-fd, write-fd.

• in socat V2, the order left-to-right / right-to-left (which correlates to write / read) is preferred because this makes a bidirectional gzip equivalent to gzip / gunzip

• to prevent semantic mismatch, the dual address separator was changed from “!!” to “%” (so you will get an error instead of malfunction after upgrade)

• the new equivalent to stdio is:

  socat 1%0 ...
Example: SSH through Proxy

- Problem: many corporate firewall/proxy systems do not allow SSH to port 22 or via proxy to port 443
- Solution: encapsulate SSH into SSL and HTTP-Connect.

- Server at home:
  
  ```
  socat ssl-l:443,reuseaddr,fork,cert=server.pem,verify=0 \
  exec:'/usr/sbin/sshd -i'
  ```

- Client at work:
  
  ```
  $HOME/.ssh/config:
  Host server.home.at
  ProxyCommand socat - 'ssl,verify=0|proxy-connect:%h:443|\tcp:proxy:8080'
  ```

- Now invoke ssh on client at work:

  ```
  ssh server.home.at  # or with -D 1080
  ```
socat Version 2 beta available

- Currently (June 2007) socat V2 is in beta status
- (almost) all tests are passed (V1 and chain tests)
- Problem with multiple socat engines: -v and -x dump in every engine
- Logging with -d -d shows too much stuff
- Documentation is not actual
- full featured engine with tracing, EOL conversion, ignoreeof etc. not good in all places
- ...
- socat 1.6 will be maintained for some more time
socat Chains - TODO

- Inter address for running program:
  `...|exec:myscript-with-four-fds|...`

- Same for dual:
  `...|exec:unix2dos%exec:dos2unix|...`

- Defined interface/API for easy contribution of inter addresses from community
Future – XIO API, Preload

- provide API (xio):
  ```c
  xfd = xioopen("ssl|open:/dev/ttyS0");
  bytes = xioread(xfd, buff, buflen);
  write(stdout, buff, bytes);
  xioclose(xfd);
  ```

- preload library:
  ```bash
  LD_PRELOAD=libxio.so firefox ...
  ```
  needs configuration by file or environment

- problems are with fork, signals, select(), blocking...
Contact

- Author: Gerhard Rieger
- Web page: http://www.dest-unreach.org/socat
- eMail: socat@dest-unreach.org
Links

- socat:  http://www.dest-unreach.org/socat
          http://www.dest-unreach.org/socat/doc
- download:  http://www.dest-unreach.org/socat/download/
- GNU netcat:  http://netcat.sourceforge.net/
- sbd:  http://tigerteam.se/software_en.shtml
- ncat:  http://sourceforge.net/projects/nmap-ncat/
- TOR:  http://tor.eff.org/index.html
- gender changer page:
          http://www.csnc.ch/static/download/misc/TCP-IP_GenderChanger_CSNC_V1.0.pdf