Mark Rasch, SecurityFocus, 2004.10.25:

"The latest tool in competition: hacking

"Your competitor has a wildly successful Web-based tool which is being used by many of your customers. Do you (A) give up and get out of the business; (B) set up a team of product developers to make a competing product; or (C) hack into the competitor's website, steal the code, and for good measure hire their critical employees to develop an exact duplicate of their website. If you answered (C) then congratulations and welcome to the new world of competitive hacking.

"On October 15, the United States Court of Appeals for the Ninth Circuit in Seattle, Washington had to deal with the case of two competing websites geared at helping long-distance truckers take on additional revenue-producing load to avoid the unprofitable practice of 'dead-heading'—driving a truck that was less than full. One company, Creative Computing, created a successful website called Truckstop.com to help match truckers with loads. In the words of the court, a second company, Getloaded.com, 'decided to compete, but not honestly.'

"Getloaded.com used many mechanisms to acquire data from the Truckstop.com

website. Initially, they just copied the most current lists of unmatched drivers and loads. When Truckstop started using user IDs and passwords, Getloaded did the same. Reasoning correctly that truckers using both sites would create the same userid's and passwords, Getloaded officials logged into Truckstop's site using their customers' IDs. Then they registered a defunct company as a subscriber as another route to getting access to the data.

"But this wasn't enough. As the court of appeals noted, 'Getloaded's officers also hacked into the code Creative used to operate its website. Microsoft had

distributed a patch to prevent a hack it had discovered, but Creative Computing had not yet installed the patch on truckstop.com. Getloaded's president and vice-president hacked into Creative Computing's website through the back door that this patch would have locked.'

...

"Getloaded also 'hired away a Creative Computing employee who had given Getloaded an unauthorized tour of the truckstop.com website,' the court noted. 'This employee, while still working for Creative, accessed confidential information regarding several thousand of Creative's customers. He downloaded, and sent to

his home e-mail account, the confidential address to truckstop.com's server so that he could access the server from home and retrieve customer lists.' "

File descriptors

Each process has an array of
file descriptors inside system data.

Some important components of each fd:

- "readable": 1 if the file is open for reading, 0 otherwise;
- "writable": 1 if the file is open for writing, 0 otherwise;
- "type": 1 for disk file, 2 for network socket, 3 for pipe, etc.
- "inode number": where the file is located on disk, if it's a disk file;
- "position": location in file of next byte to read or write; ...

Details: `/usr/include/sys/file.h`, `/usr/include/sys/socket.h`, et al.

Programs refer to file descriptors
through array indices: 0, 1, 2, 3, etc.

`read(3,buf,10)` syscall
reads 10 bytes from the file
that descriptor 3 refers to.

`write(3,buf,10)` syscall
write 10 bytes to the file.

`close(3)` syscall clears descriptor 3:
not readable, not writable, etc.

`open("/etc/passwd",O_RDONLY)` syscall
searches for first array index
where descriptor is closed;
sets up that descriptor
reading from `/etc/passwd`;
and returns the array index.

e.g. Process with no files open:

	read	write	type	inode	pos
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

`open("/etc/passwd",O_RDONLY)`

returns 0:

	read	write	type	inode	pos
0	1	0	1	3650113	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

`open("/home/joe/copy",O_WRONLY)`
returns 1:

	read	write	type	inode	pos
0	1	0	1	3650113	0
1	0	1	1	1647555	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

`open("/home/joe/errorlog",O_WRONLY)`
returns 2:

	read	write	type	inode	pos
0	1	0	1	3650113	0
1	0	1	1	1647555	0
2	0	1	1	1647590	0
3	0	0	0	0	0
4	0	0	0	0	0

`open("/home/joe/copystatus",O_RDWR)` returns 3:

	read	write	type	inode	pos
0	1	0	1	3650113	0
1	0	1	1	1647555	0
2	0	1	1	1647590	0
3	1	1	1	1647584	0
4	0	0	0	0	0

`read(0,buf,10)` returns 10:

	read	write	type	inode	pos
0	1	0	1	3650113	10
1	0	1	1	1647555	0
2	0	1	1	1647590	0
3	1	1	1	1647584	0
4	0	0	0	0	0

`write(1,buf,7)` returns 7:

	read	write	type	inode	pos
0	1	0	1	3650113	10
1	0	1	1	1647555	7
2	0	1	1	1647590	0
3	1	1	1	1647584	0
4	0	0	0	0	0

`read(0,buf,5)` returns 5:

	read	write	type	inode	pos
0	1	0	1	3650113	15
1	0	1	1	1647555	7
2	0	1	1	1647590	0
3	1	1	1	1647584	0
4	0	0	0	0	0

`execve` does not clear fds.

C library functions expect
program to be started with
descriptor 0 reading something,
descriptor 1 writing something,
descriptor 2 writing something.

0 is "standard input" (`stdin`);
1 is "standard output" (`stdout`);
2 is "standard error" (`stderr`).

Typical program reads from `stdin`,
writes to `stdout`, complains to `stderr`.
Can run program to read from user,
from a disk file, from network, etc.,
by setting `stdin` before `execve`.

e.g. UNIX `sort` program
reads lines of input
using `read(0,...)`.
Prints same lines in order
using `write(1,...)`.

Can run

```
sort < data.in > data.out
```

to have `sort` program
read `data.in`, write `data.out`.
`/bin/sh` opens `data.in` and `data.out`
before `execve("/usr/bin/sort",...)`.

Can run

```
tcpserver 0 10000 sort
```

to have `sort` program
talk to network connections.