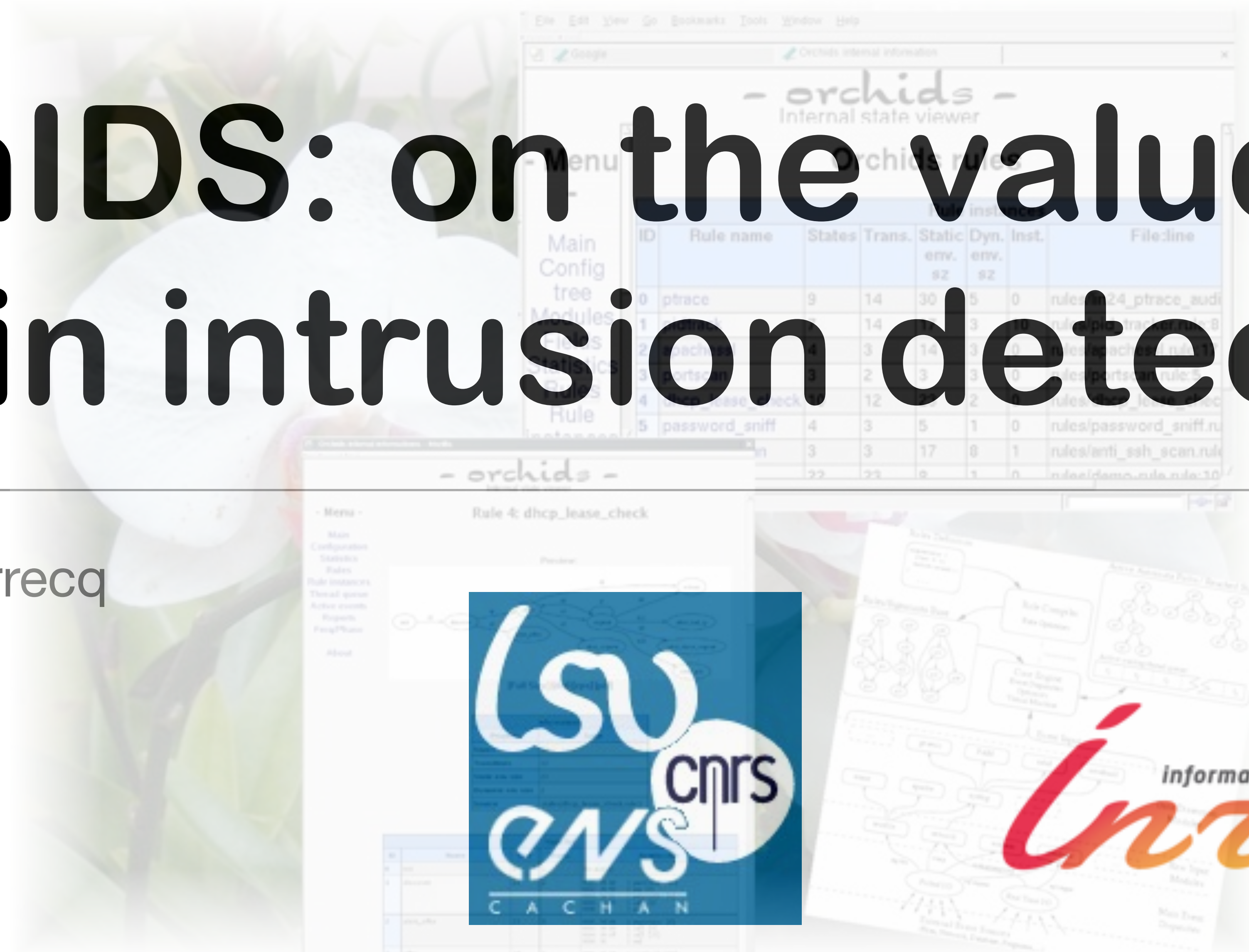
OrchIDS: on the value of rigor in intrusion detection



Jean Goubault-Larrecq

CPS, Grenoble, July 08 2014

Outline

1. A few **scary stories** about computer security

2. **ORCHIDS**: an intrusion prevention system

3. **Semantics** and algorithms

4. **NetEntropy**: detecting subverted cryptographic flows

5. Conclusion

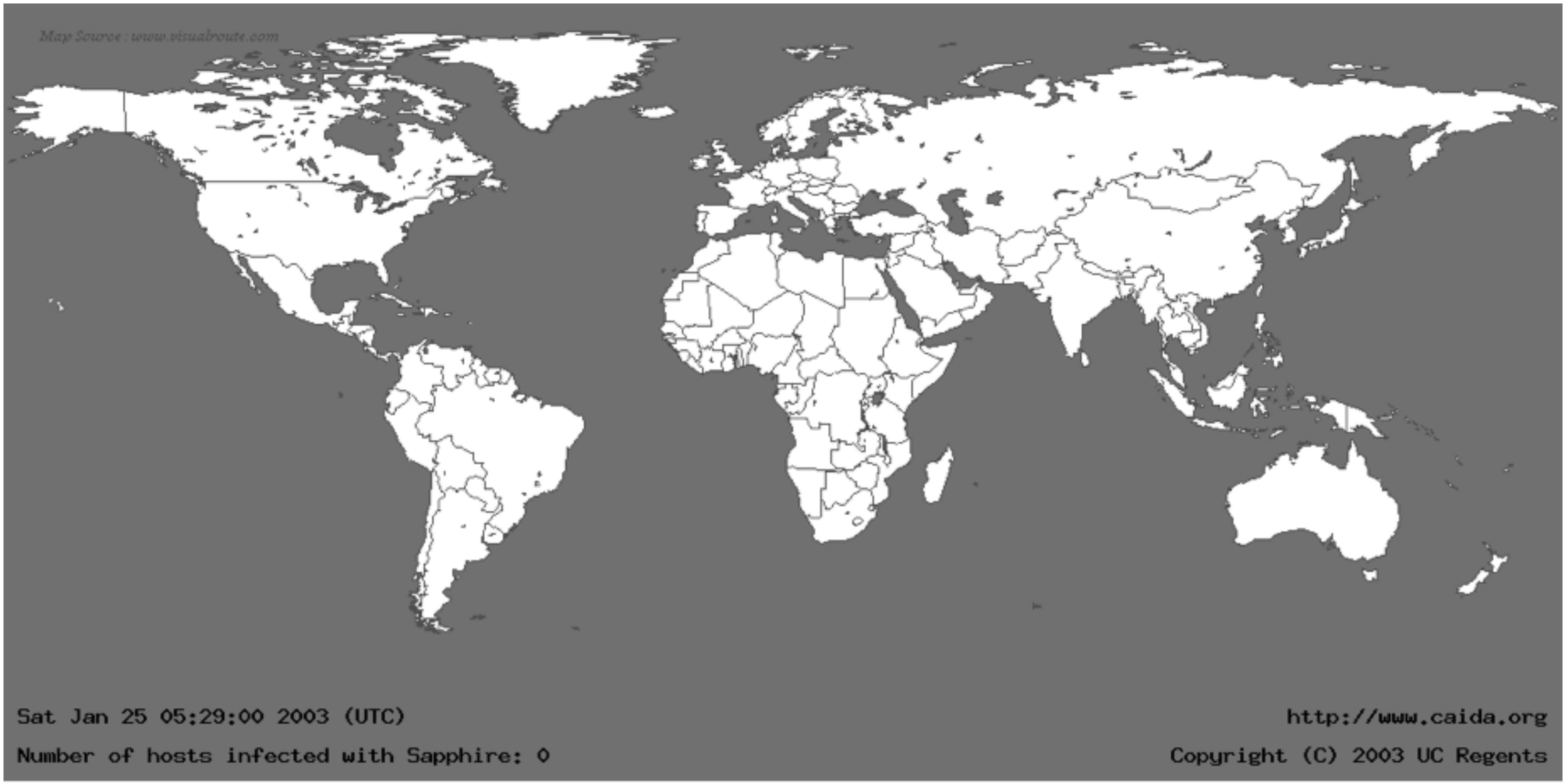
Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

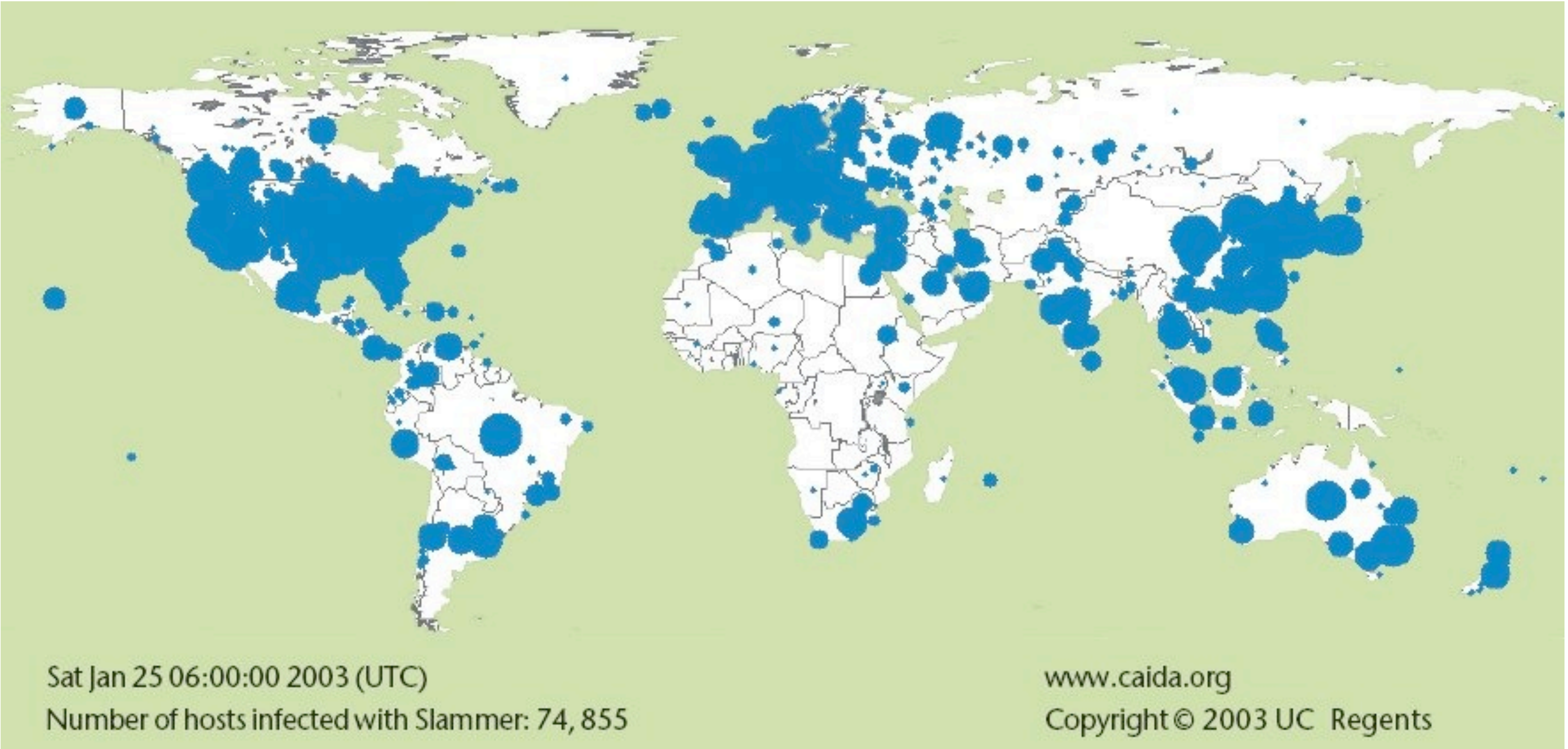
Example 1: Slammer (2003)

- An internet **worm** designed to propagate quickly
- which did not do anything...
- ... except propagate ...
- ... and bring networks to their knees

Slammer: Jan. 25, 2003, 05:29



Slammer: Jan. 2003, 06:00



Slammer: impact

- **911 emergency number** in Seattle: down
- **Canceled flights** Newark hub, Continental Airlines
- **Internet down** in Portugal, South Korea
- **No mobile phone service**, South Korea
- 5 out of the 13 Internet **backbone servers** down
- Estimated cost: **> $ 1 billion**

Slammer: impact

PRINT EMAIL COMMENT

Slammer worm crashed Ohio nuke plant network

Kevin Poulsen, SecurityFocus 2003-08-19

The Slammer worm penetrated a private computer network at Ohio's Davis-Besse nuclear power plant in January and disabled a safety monitoring system for nearly five hours, despite a belief by plant personnel that the network was protected by a firewall, SecurityFocus has learned.

The breach did not pose a safety hazard. The troubled plant had been offline since February, 2002, when workers discovered a 6-by-5-inch hole in the plant's reactor head. Moreover, the monitoring system, called a Safety Parameter Display System, had a redundant analog backup that was unaffected by the worm. But at least one expert says the case illustrates a growing cybersecurity problem in the nuclear power industry, where interconnection between plant and corporate networks is becoming more common, and is permitted by federal safety regulations.

The Davis-Besse plant is operated by FirstEnergy Corp., the Ohio utility company that's become the focus of an investigation into the northeastern U.S. blackout last week.

The incident at the plant is described in an April e-mail to the Nuclear Regulatory Commission (NRC) from FirstEnergy, and in a similarly-worded March safety advisory distributed privately throughout the industry over the "Nuclear Network," an information-sharing program run by the Institute of Nuclear Power Operations. The March advisory was issued to "alert the industry to consequences of Internet Worms and Viruses on Plant Computer Systems."

Slammer: impact

News
Infocus
- Foundations
- Microsoft
- Unix
- IDS
- Incidents
- Virus
- Pen-Test
- Firewalls

Columnists
Mailing Lists
- Newsletters
- Bugtraq
- Focus on IDS
- Focus on Linux
- Focus on Microsoft
- Forensics
- Pen-test
- Security Basics
- Vuln Dev

Vulnerabilities
Jobs
- Job Opportunities
- Resumes
- Job Seekers
- Employers

Tools
RSS
- News
- Vulns

Security Research

PRINT EMAIL COMMENT

Slammer worm crashed Ohio nuke plant network

Kevin Poulsen, SecurityFocus 2003-08-19

The Slammer worm penetrated a private computer network at Ohio's Davis-Besse nuclear power plant in January and disabled a safety monitoring system for nearly five hours, despite a belief by plant personnel that the network was protected by a firewall, SecurityFocus has learned.

The breach did not pose a safety hazard. The troubled plant had been offline since February, 2002, when workers discovered a 6-by-5-inch hole in the plant's reactor head. Moreover, the monitoring system, called a Safety Parameter Display System, had a redundant analog backup that was unaffected by the worm. But at least one expert says the case illustrates a growing cybersecurity problem in the nuclear power industry, where interconnection between plant and corporate networks is becoming more common, and is permitted by federal safety regulations.

The Davis-Besse plant is operated by FirstEnergy Corp., the Ohio utility company that's become the focus of an investigation into the northeastern U.S. blackout last week.

The incident at the plant is described in an April e-mail to the Nuclear Regulatory Commission (NRC) from FirstEnergy, and in a similarly-worded March safety advisory distributed privately throughout the industry over the "Nuclear Network," an information-sharing program run by the Institute of Nuclear Power Operations. The March advisory was issued to "alert the industry to consequences of Internet Worms and Viruses on Plant Computer Systems."

Anatomy of the beast

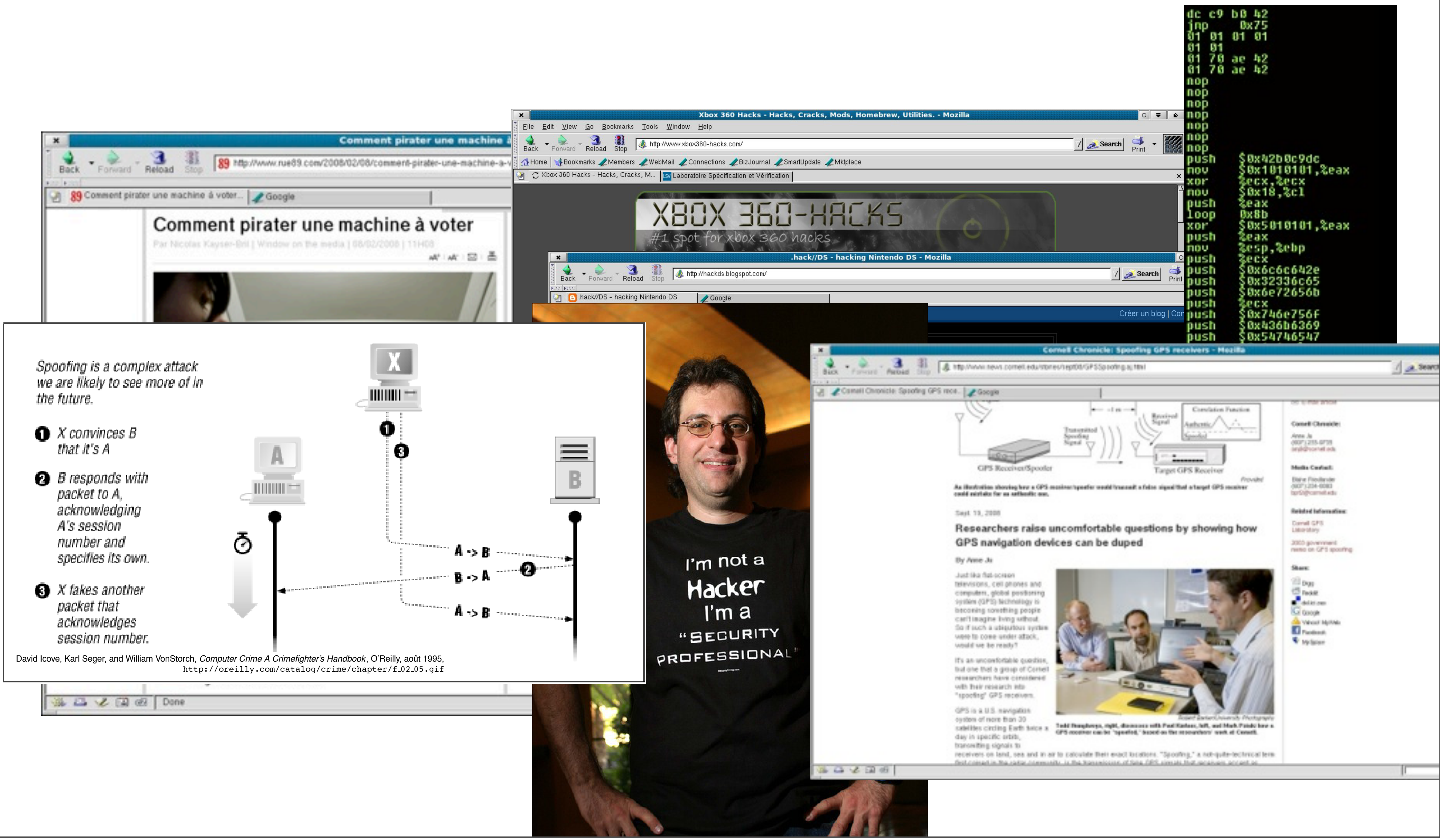
```
04
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
01 01 01 01
```

```
dc c9 b0 42
jmp     0x75
01 01 01 01
01 01
01 70 ae 42
01 70 ae 42
nop
nop
nop
nop
nop
nop
nop
push    $0x42b0c9dc
mov     $0x1010101,%eax
xor     %ecx,%ecx
mov     $0x18,%cl
push    %eax
loop    0x8b
xor     $0x5010101,%eax
push    %eax
mov     %esp,%ebp
push    %ecx
push    $0x6c6c642e
push    $0x32336c65
push    $0x6e72656b
push    %ecx
push    $0x746e756f
push    $0x436b6369
push    $0x54746547
mov     $0x6c6c,%cx
push    %ecx
push    $0x642e3233
push    $0x5f327377
mov     $0x7465,%cx
push    %ecx
push    $0x6b636f73
mov     $0x6f74,%cx
push    %ecx
push    $0x646e6573
mov     $0x42ae1018,%esi
lea     0xffffffd4(%ebp),%eax
push    %eax
call    *(%esi)
push    %eax
lea     0xffffffe0(%ebp),%eax
push    %eax
lea     0xfffffff0(%ebp),%eax
push    %eax
call    *(%esi)
push    %eax
mov     $0x42ae1010,%esi
mov     (%esi),%ebx
mov     (%ebx),%eax
cmp     $0x51ec8b55,%eax
je      0x105
mov     $0x42ae101c,%esi
call    *(%esi)
```

- Terribly small: 376 bytes
- Does nothing... except propagate
- Took networks down, worldwide, by flooding them with copies of itself (Denial of Service)

Paul Boutin, *Slammed!*, WiReD magazine 11.07, July 2003, http://www.wired.com/wired/archive/11.07/slammer.html

Computer (in)security



David Icove, Karl Seger, and William VonStorch, *Computer Crime A Crimefighter's Handbook*, O'Reilly, août 1995, http://oreilly.com/catalog/crime/chapter/f_02_05.gif

Computer (in)security

http://www.docstoc.com/docs/22073608/Estonia-cyber-attacks-2007

Estonia cyber attacks 2007

- Known as the Estonian Cyberwar

Cyber War 2.0 — Russia v. Georgia

by WARD CARROLL on AUGUST 13, 2008

http://defensetech.org/2008/08/13/cyber-war-2-0-russia-v-georgia/

J'aime 3 personnes aiment ça. Inscription pour voir ce que vos amis aiment.

The second real cyber was has broken out. On August 8th, Russian troops crossed into South Ossetia vowing to defend what they called "Russian compatriots". As this was taking place, a multi-faceted cyber attack began against the Georgian infrastructure and key government web sites. The attack modalities included: Defacing of Web Sites (Hacktivism), Web-based Psychological Operations (Psyc-Ops), a fierce propaganda campaign (PC) and of

Massive Cyber Attacks Uncovered

Feb. 19, 2010, http://www.darkgovernment.com/news/massive-cyber-attacks-uncovered/

More than 75,000 computer systems at nearly 2,500 companies in the United States and around the world have been hacked in what appears to be one of the largest and most sophisticated attacks by cyber criminals discovered to date, according to a northern Virginia security firm.

The attack, which began in late 2008 and was discovered last month, targeted proprietary corporate data, e-mails, credit-card transaction data and login credentials at companies in the health and technology industries in 196 countries, according to Herndon-based NetWitness.

News of the attack follows reports last month that the computer networks at Google and more than 30 other large financial, energy, defense, technology and media firms had been compromised. Google said the attack on its system originated in China.

This latest attack does not appear to be linked to the Google intrusion, said Amit Yoran, NetWitness's chief executive. But it is significant, he said, in its scale and in its apparent demonstration that the criminal groups' sophistication in cyberattacks is approaching that of nation states such as China and Russia.

Série d'attaques informatiques contre le gouvernement israélien

http://www.radio-canada.ca/nouvelles/International/2013/04/07/002-anonymous-attaques-israel.shtml

Mise à jour le dimanche 7 avril 2013 à 8 h 30 HAE | Radio-Canada avec Agence France-Presse

Commenter 41 | +1 1 | Recommander 194 | Tweet 34 | Partager

Cyberattaque contre Israël

July 17, 2012

STUXNET: ANATOMY OF THE FIRST WEAPON MADE ENTIRELY OUT OF CODE

http://socks-studio.com/2012/07/17/stuxnet-anatomy-of-the-first-weapon-made-entirely-out-of-code/

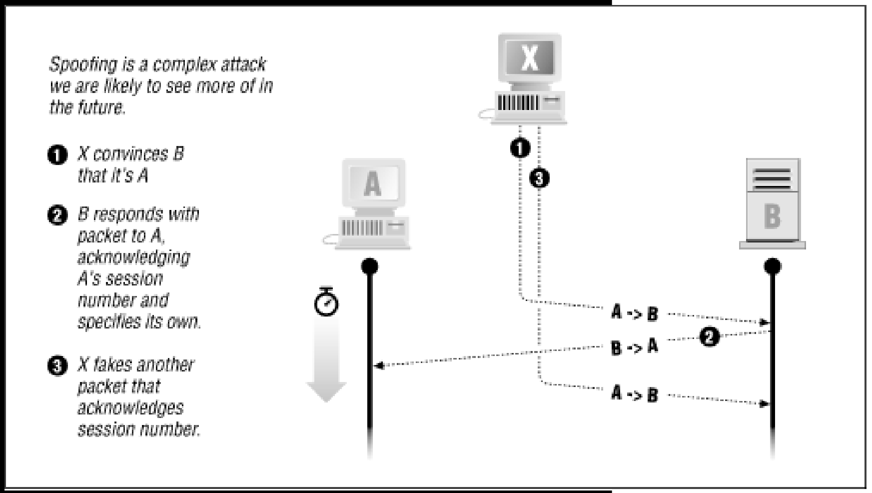
by fosco lucarelli politics, psychogeographies, technology, virtual chronicles, world weird itself

Stuxnet is the first computer virus (precisely a "worm") created to target, study, infect and subvert only industrial systems, namely Siemens'.

The Mitnick Attack (1994)

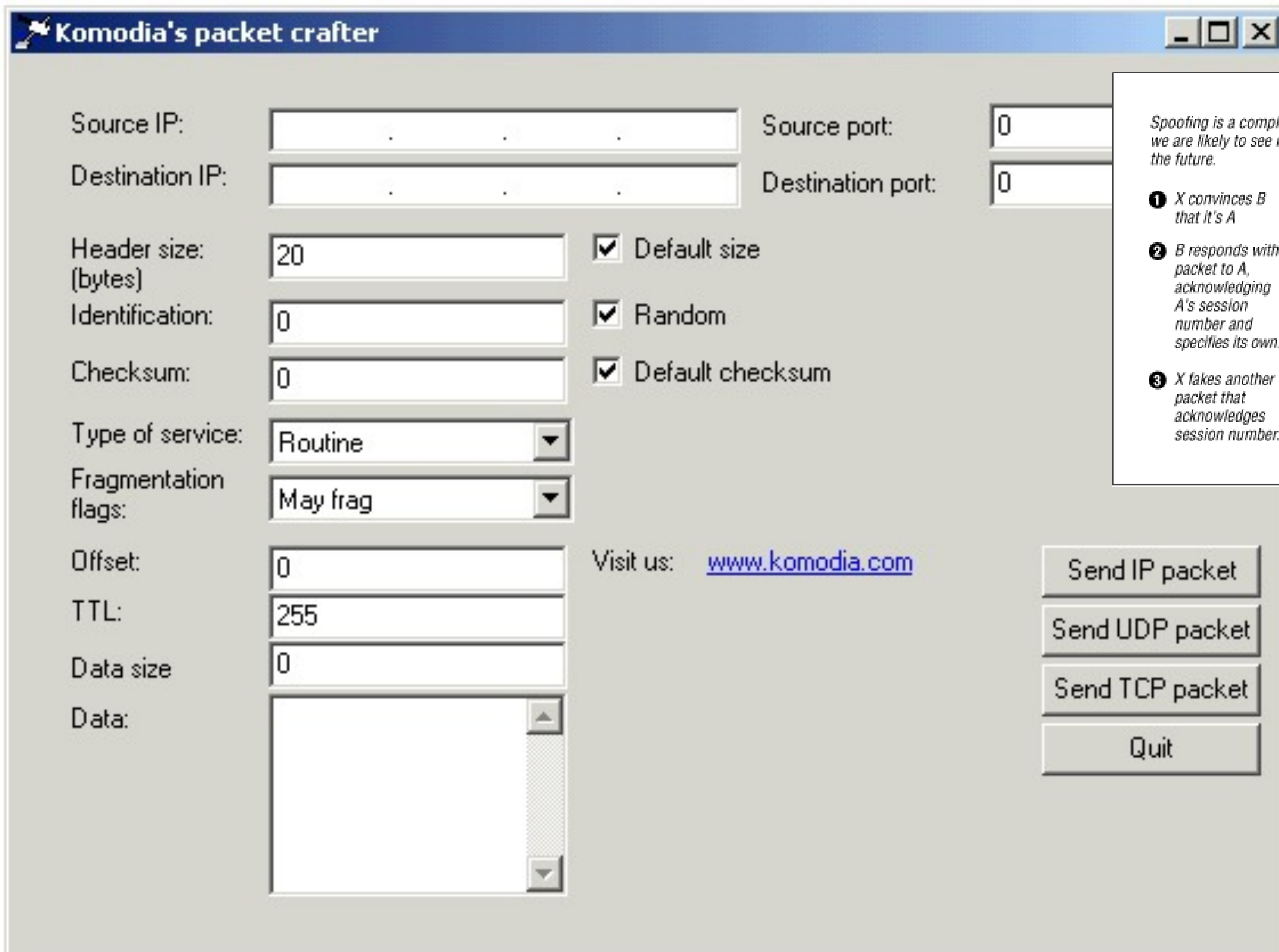
Easy! (for an expert)

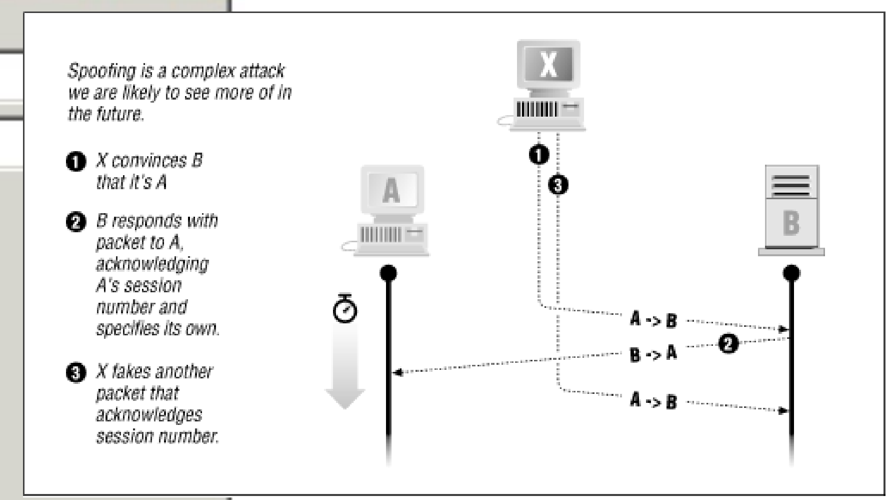
```
14:09:32 toad.com# finger -l @ARIEL
14:10:21 toad.com# finger -l @RIMMON
14:10:50 toad.com# finger -l root@RIMMON
14:11:07 toad.com# finger -l @OSIRIS
14:11:38 toad.com# showmount -e OSIRIS
14:11:49 toad.com# rpcinfo -p OSIRIS
14:12:05 toad.com# finger -l root@OSIRIS

...

14:18:37 [root@apollo /tmp]#rsh OSIRIS "echo + + >>/.rhosts"
```



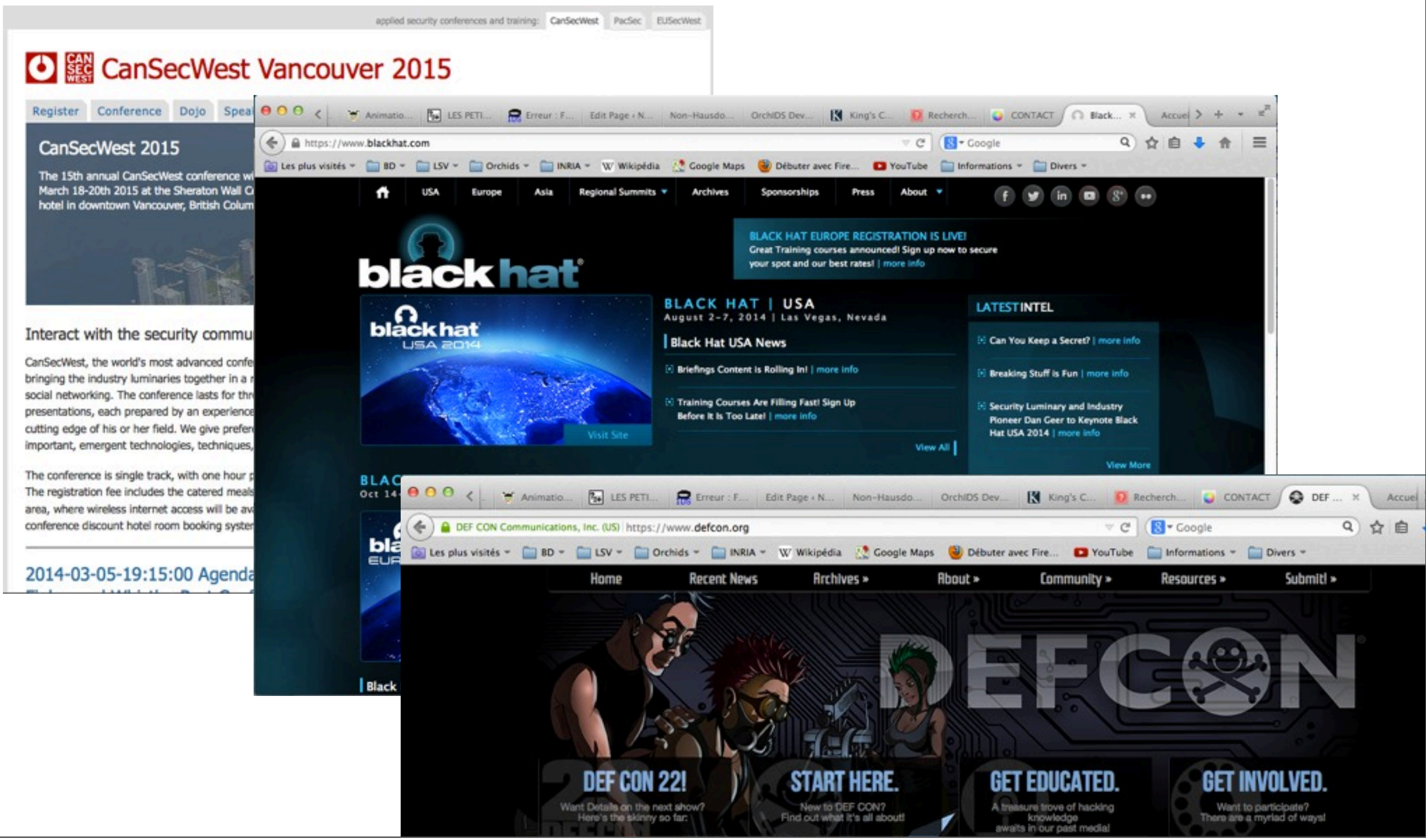
The Mitnick Attack (in 2009)

Using off-the-shelf software, e.g.:

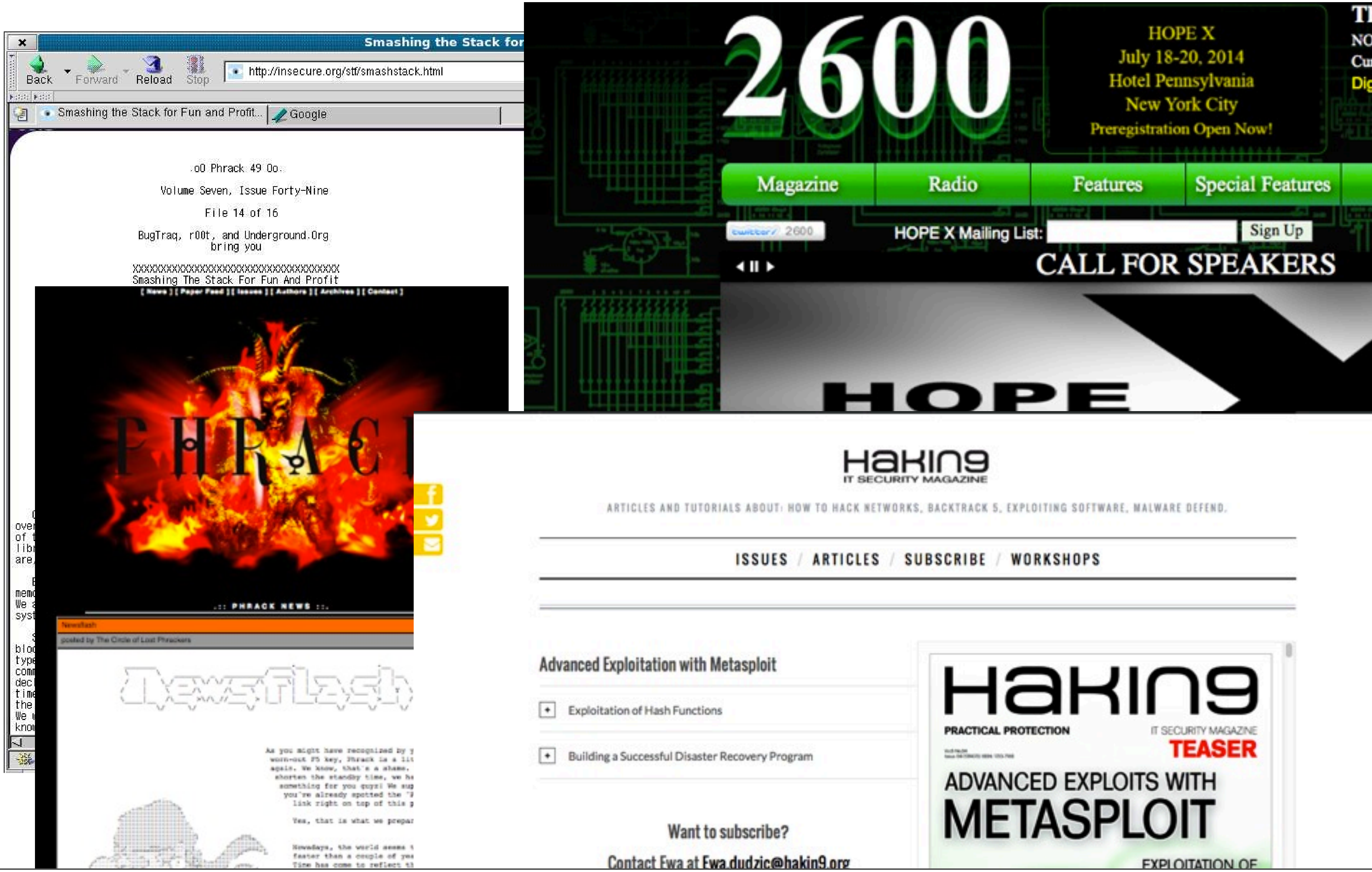




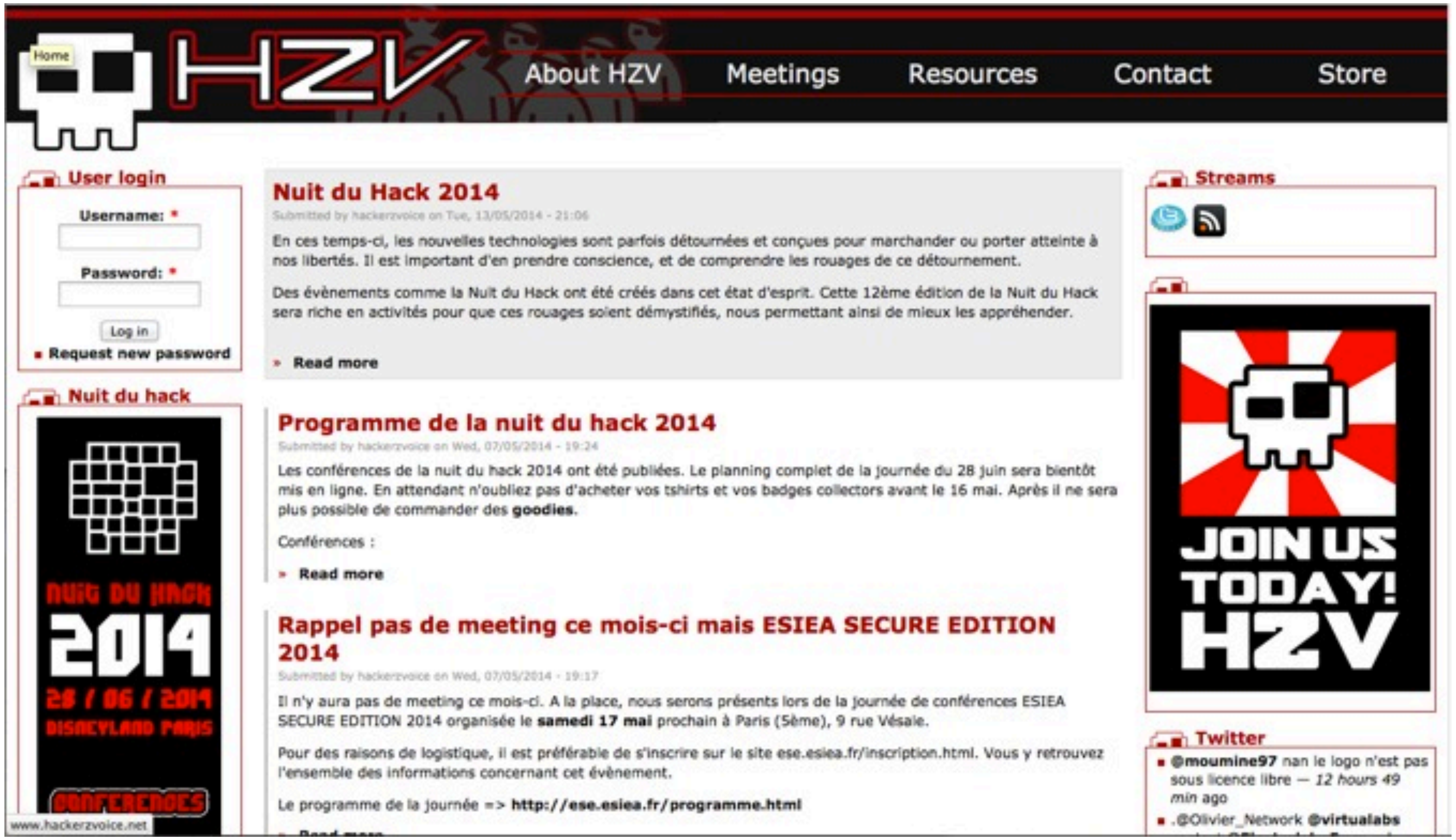
International conferences



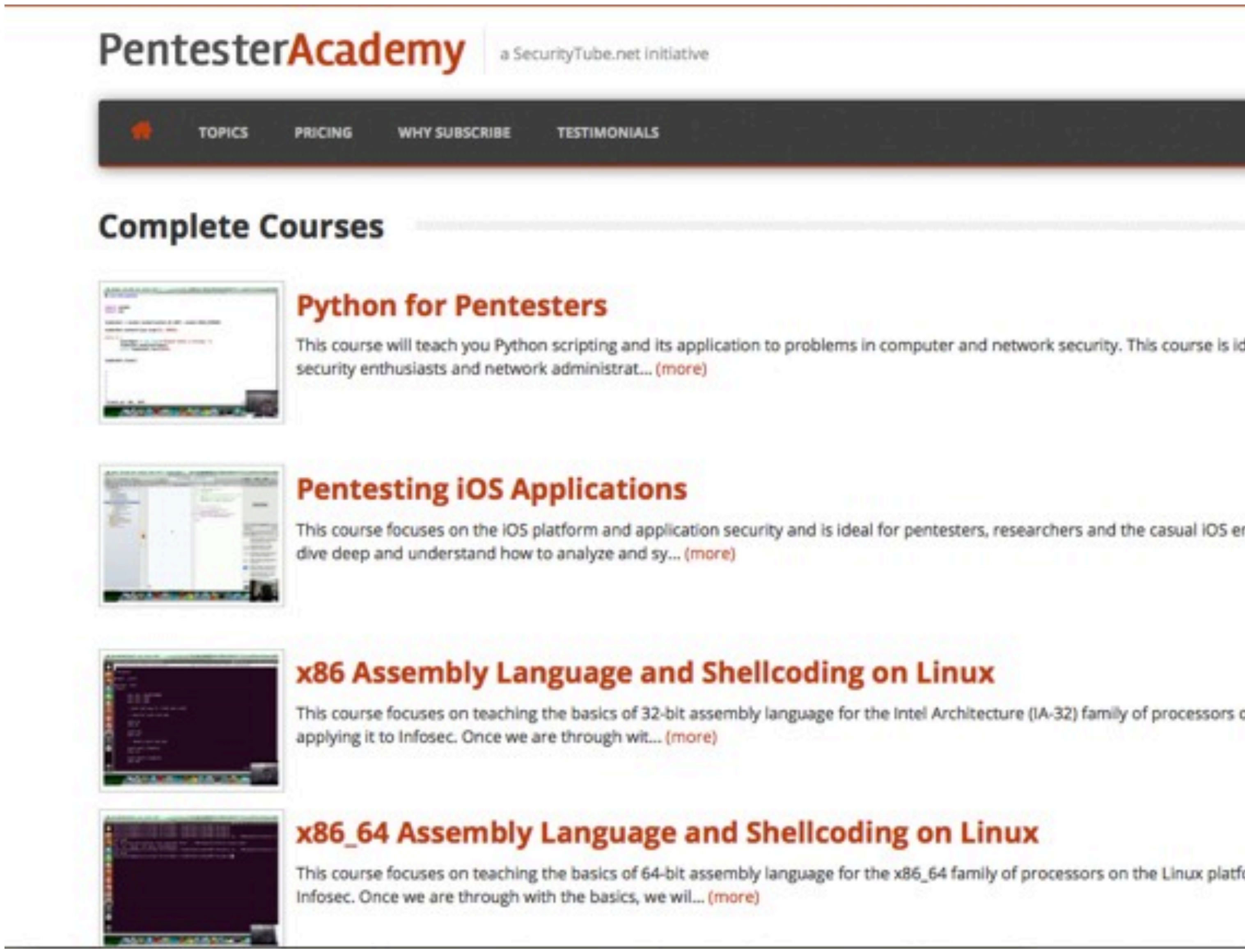
On-line journals

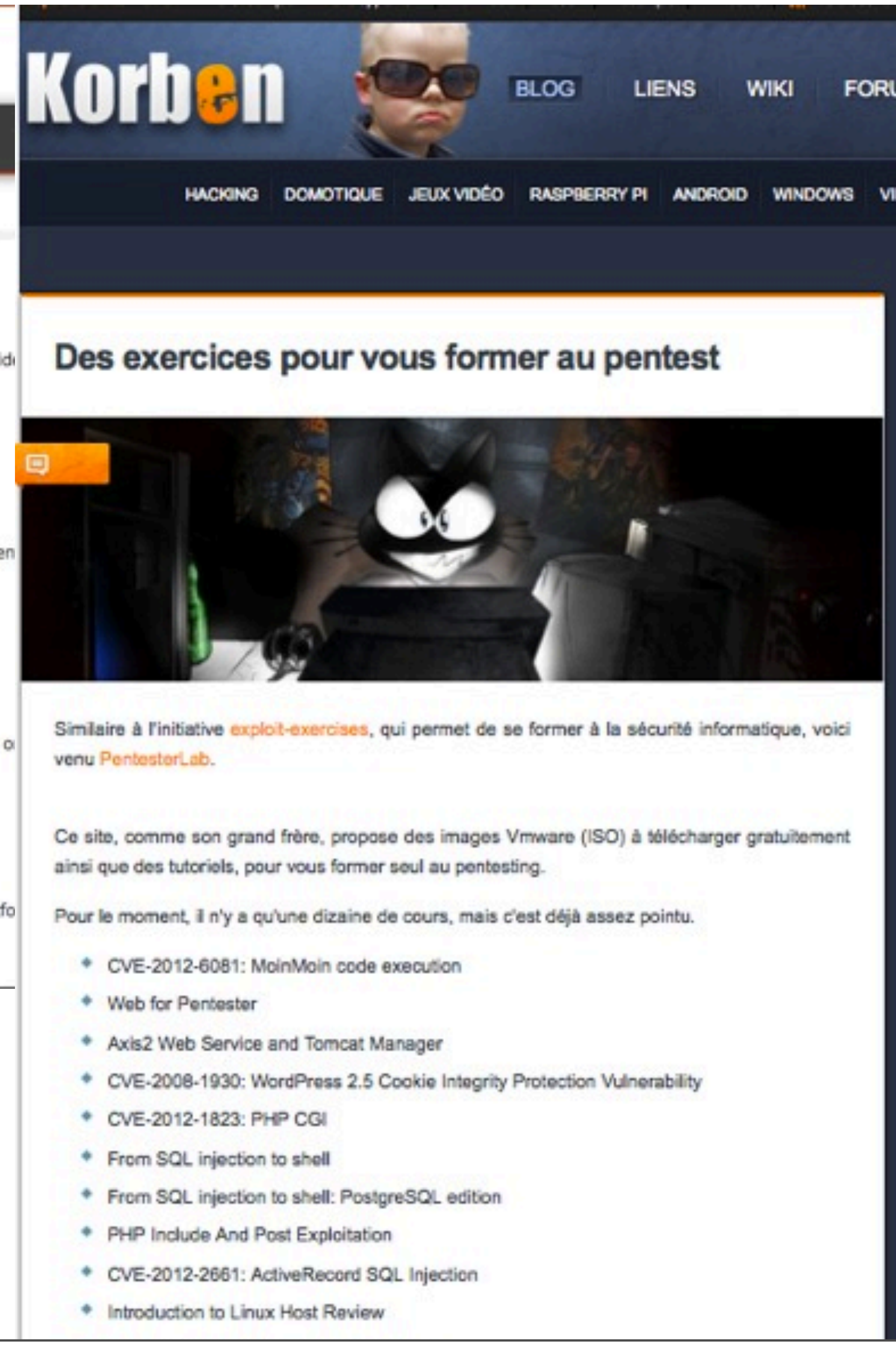


Also en français

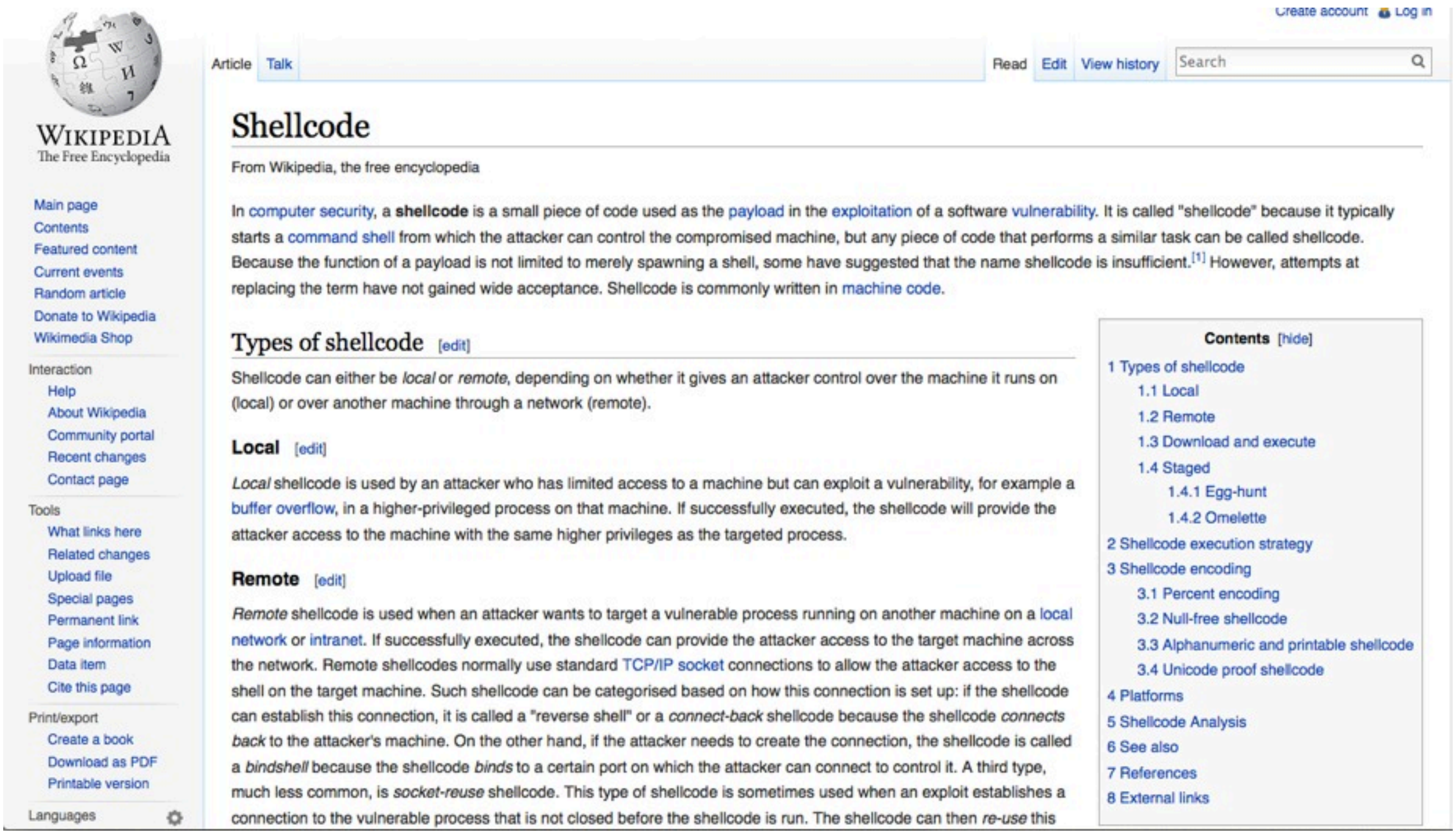


On-line courses





Google, Wikipedia are your friends



Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Data item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages

Shellcode

From Wikipedia, the free encyclopedia

In computer security, a **shellcode** is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode. Because the function of a payload is not limited to merely spawning a shell, some have suggested that the name shellcode is insufficient.[1] However, attempts at replacing the term have not gained wide acceptance. Shellcode is commonly written in machine code.

Contents [hide]

Types of shellcode [edit]

Shellcode can either be *local* or *remote*, depending on whether it gives an attacker control over the machine it runs on (local) or over another machine through a network (remote).

Local [edit]

Local shellcode is used by an attacker who has limited access to a machine but can exploit a vulnerability, for example a buffer overflow, in a higher-privileged process on that machine. If successfully executed, the shellcode will provide the attacker access to the machine with the same higher privileges as the targeted process.

Remote [edit]

Remote shellcode is used when an attacker wants to target a vulnerable process running on another machine on a local network or intranet. If successfully executed, the shellcode can provide the attacker access to the target machine across the network. Remote shellcodes normally use standard TCP/IP socket connections to allow the attacker access to the shell on the target machine. Such shellcode can be categorised based on how this connection is set up: if the shellcode can establish this connection, it is called a "reverse shell" or a *connect-back* shellcode because the shellcode *connects back* to the attacker's machine. On the other hand, if the attacker needs to create the connection, the shellcode is called a *bindshell* because the shellcode *binds* to a certain port on which the attacker can connect to control it. A third type, much less common, is *socket-reuse* shellcode. This type of shellcode is sometimes used when an exploit establishes a connection to the vulnerable process that is not closed before the shellcode is run. The shellcode can then *re-use* this

Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

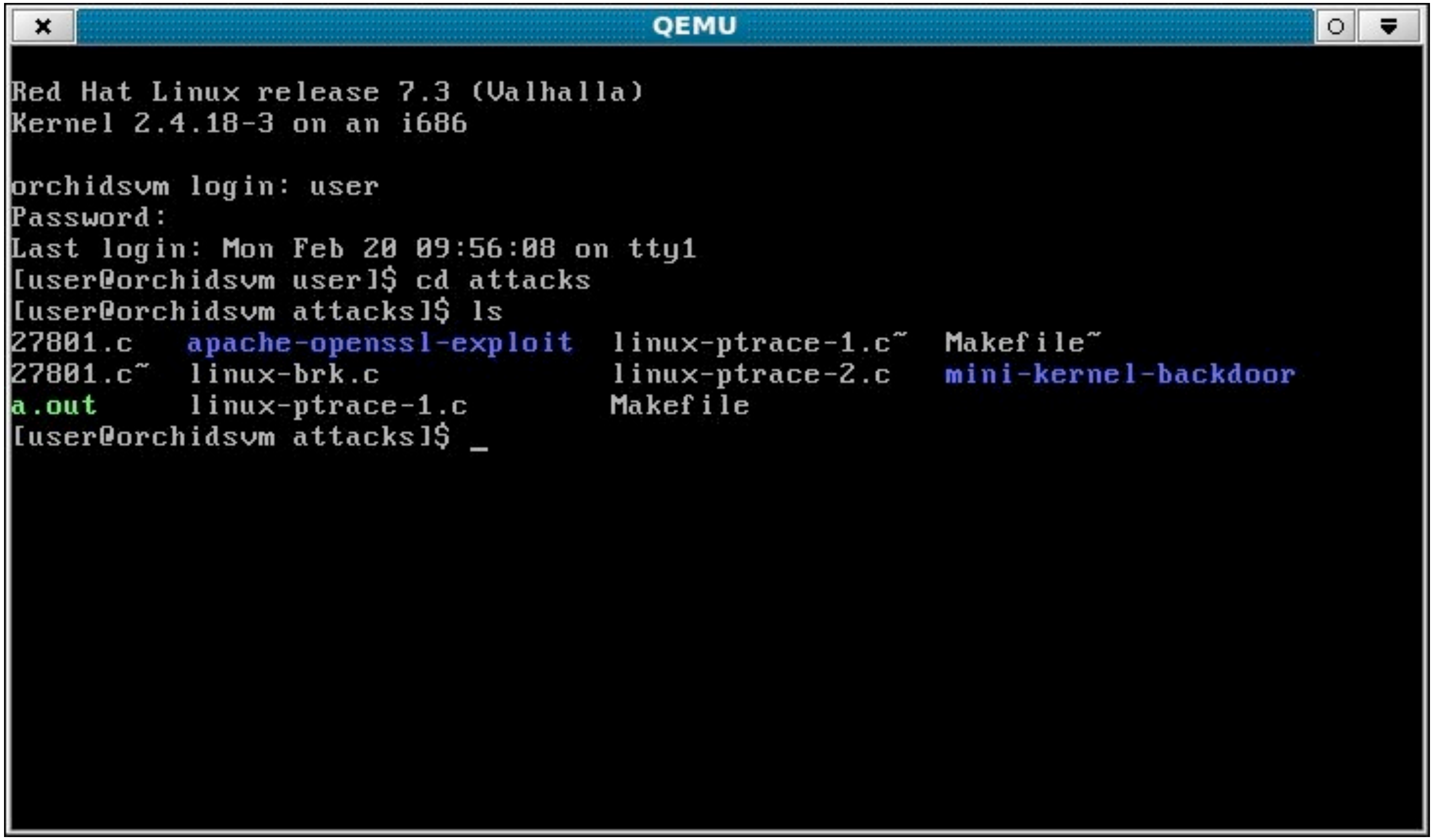
ORCHIDS

http://www.lsv.ens-cachan.fr/Software/orchids/v2.1/

The ptrace attack (Purczynski 2001, 2003): demo

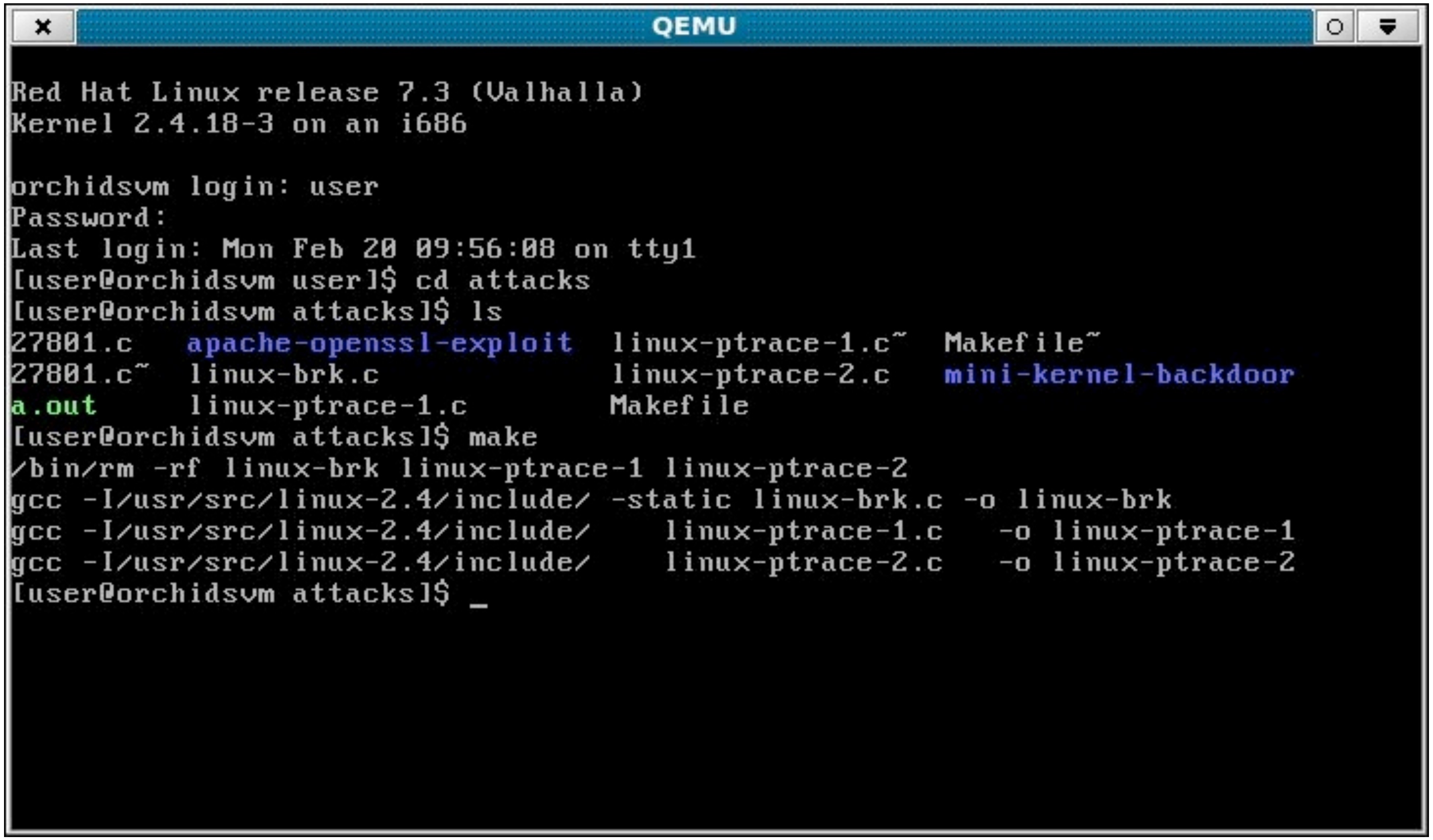
- **local-to-root** exploit
- will serve to explain some of the basic notions behind ORCHIDS

The ptrace attack (Purczynski 2001, 2003): demo



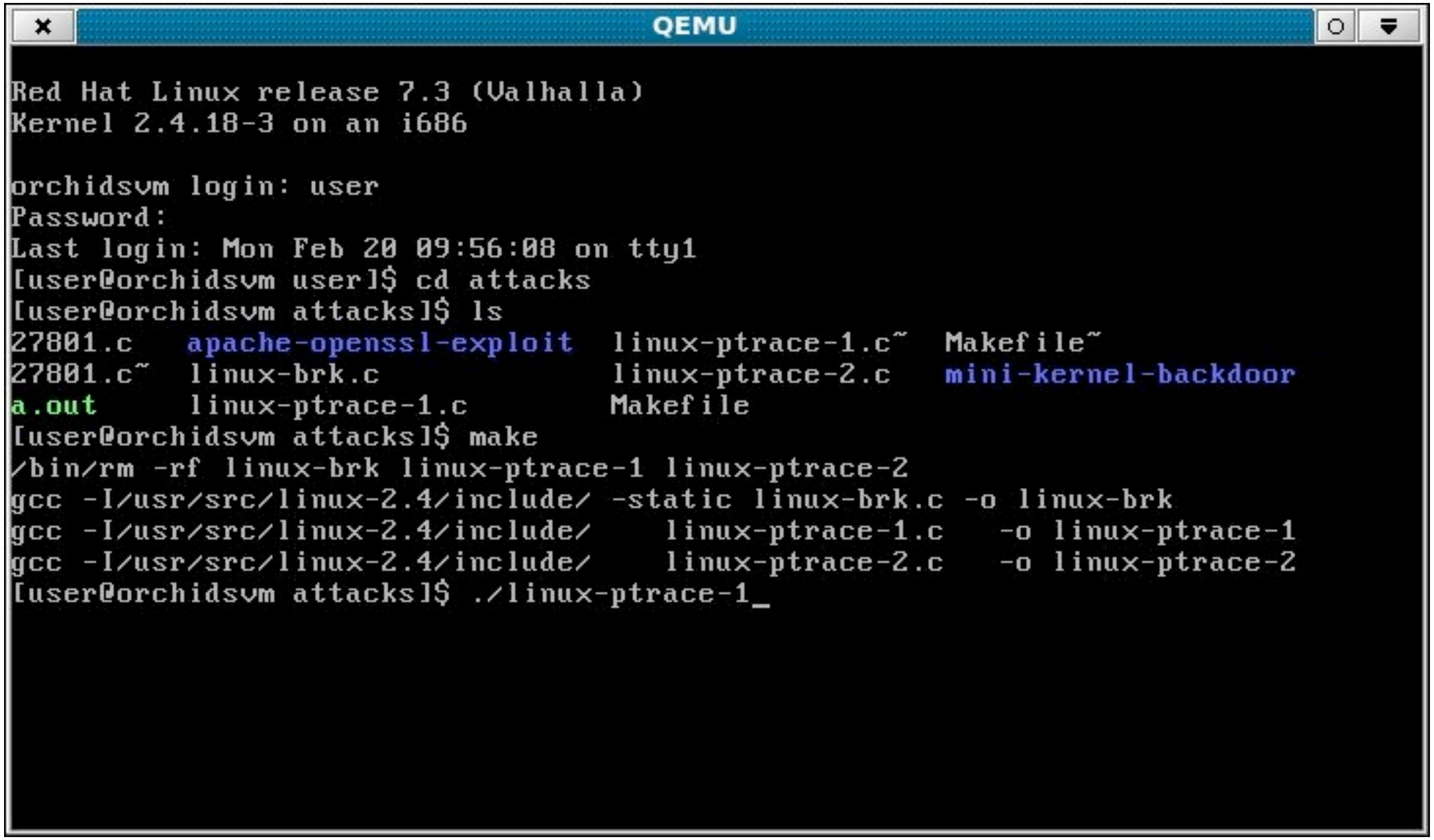
Compile attack file `linux-ptrace-1.c`....

The ptrace attack (Purczynski 2001, 2003): demo



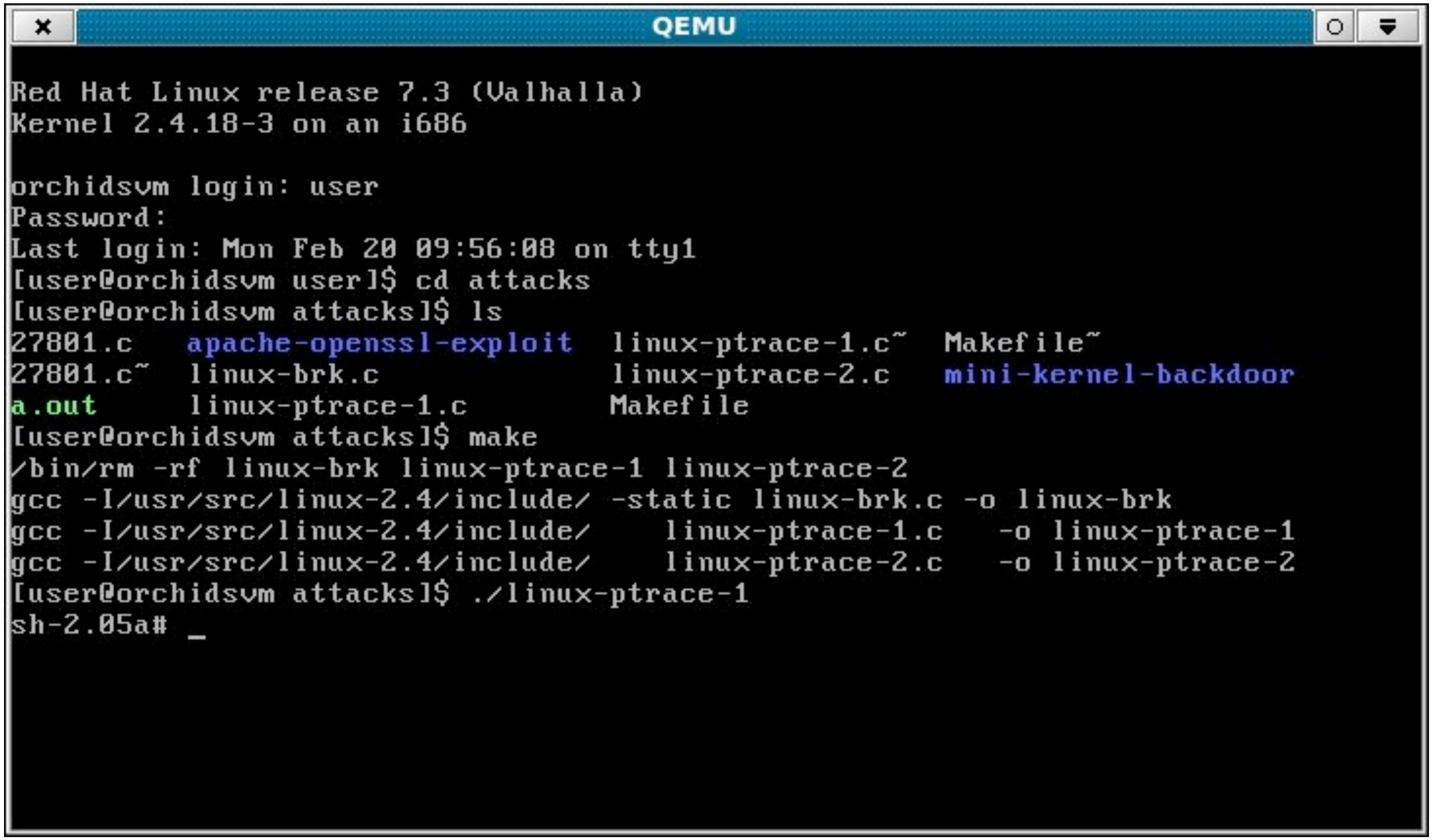
Run attack: `linux-ptrace-1`

The ptrace attack (Purczynski 2001, 2003): demo



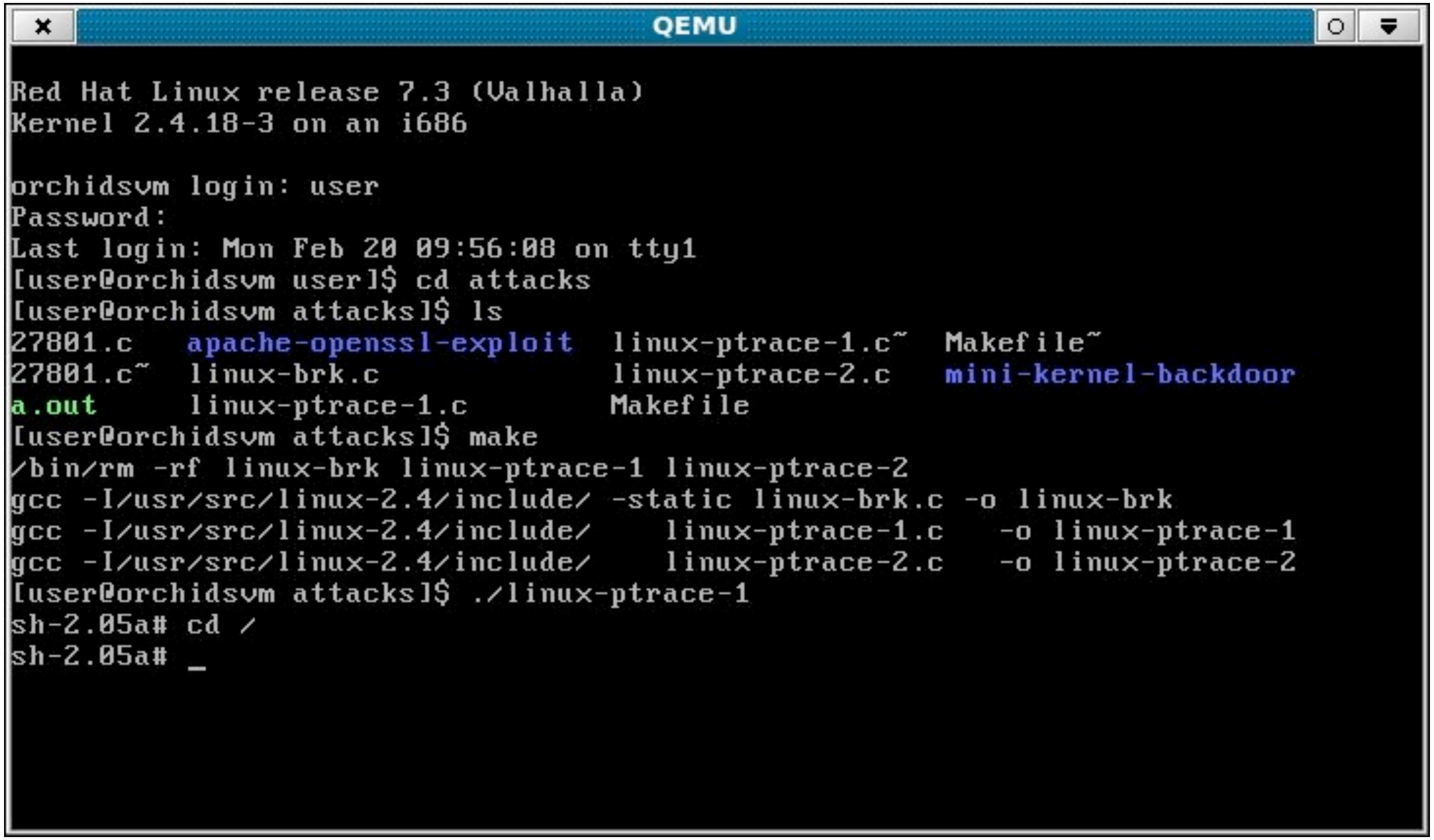
Run attack: `linux-ptrace-1`

The ptrace attack (Purczynski 2001, 2003): demo



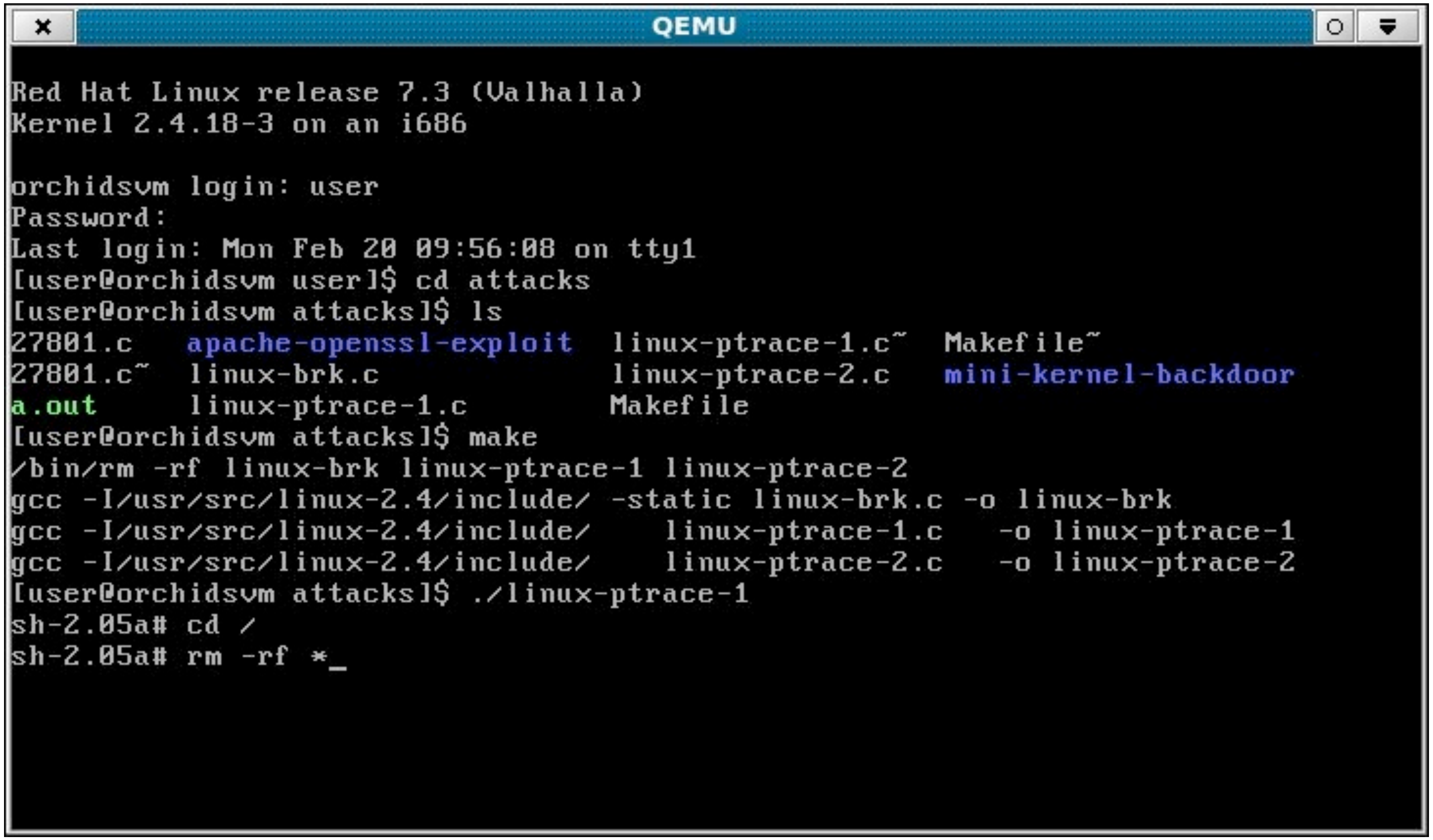
So what?

The ptrace attack (Purczynski 2001, 2003): demo



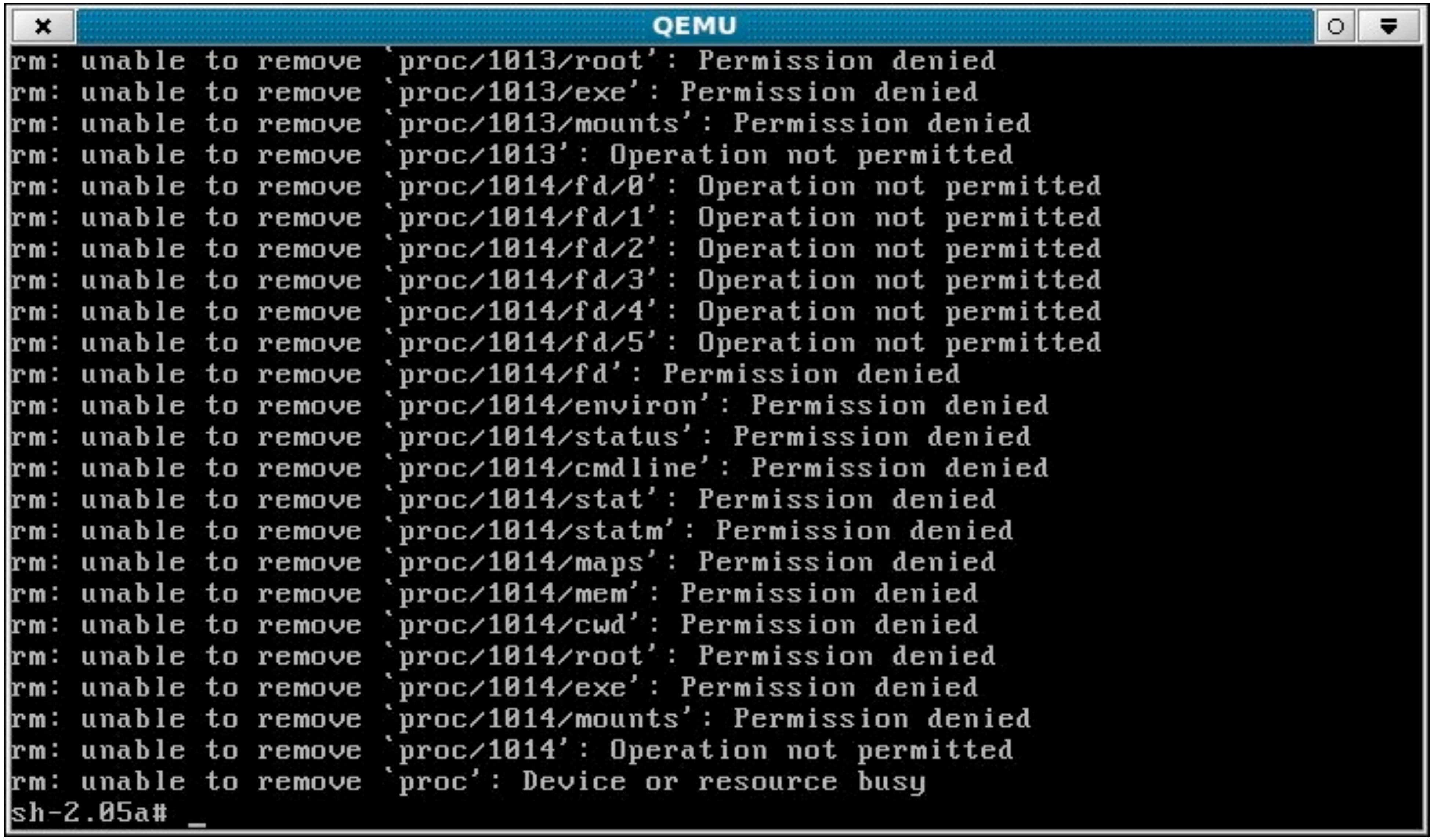
So what?

The ptrace attack (Purczynski 2001, 2003): demo



So what?

The ptrace attack (Purczynski 2001, 2003): demo



So what?

The ptrace attack (Purczynski 2001, 2003): demo

QEMU

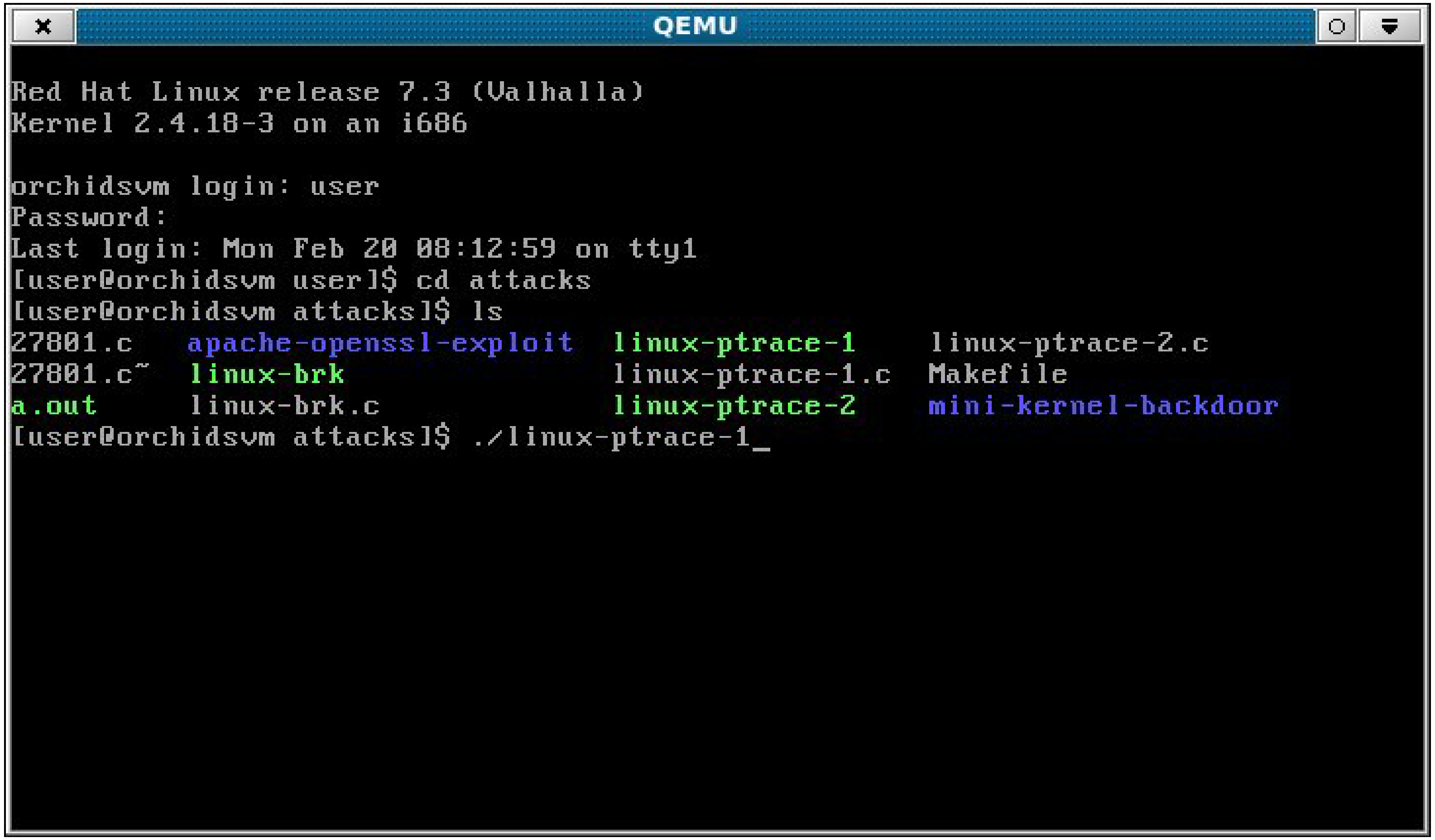
```
rm: unable to remove `proc/1013/mounts': Permission denied
rm: unable to remove `proc/1013': Operation not permitted
rm: unable to remove `proc/1014/fd/0': Operation not permitted
rm: unable to remove `proc/1014/fd/1': Operation not permitted
rm: unable to remove `proc/1014/fd/2': Operation not permitted
rm: unable to remove `proc/1014/fd/3': Operation not permitted
rm: unable to remove `proc/1014/fd/4': Operation not permitted
rm: unable to remove `proc/1014/fd/5': Operation not permitted
rm: unable to remove `proc/1014/fd': Permission denied
rm: unable to remove `proc/1014/environ': Permission denied
rm: unable to remove `proc/1014/status': Permission denied
rm: unable to remove `proc/1014/cmdline': Permission denied
rm: unable to remove `proc/1014/stat': Permission denied
rm: unable to remove `proc/1014/statm': Permission denied
rm: unable to remove `proc/1014/maps': Permission denied
rm: unable to remove `proc/1014/mem': Permission denied
rm: unable to remove `proc/1014/cwd': Permission denied
rm: unable to remove `proc/1014/root': Permission denied
rm: unable to remove `proc/1014/exe': Permission denied
rm: unable to remove `proc/1014/mounts': Permission denied
rm: unable to remove `proc/1014': Operation not permitted
rm: unable to remove `proc': Device or resource busy
sh-2.05a# ls
sh: ls: command not found
sh-2.05a# _
```

Oops...

ORCHIDS

- A **intrusion detection/prevention** tool
- developed at LSV (ENS Cachan, INRIA, CNRS) since 2002 by: JGL, J. Olivain, B. Gourdin, N.-E. Yousfi, P.-A. Sentucq
- fast
- real-time
- on-line/off-line
- multi-sources

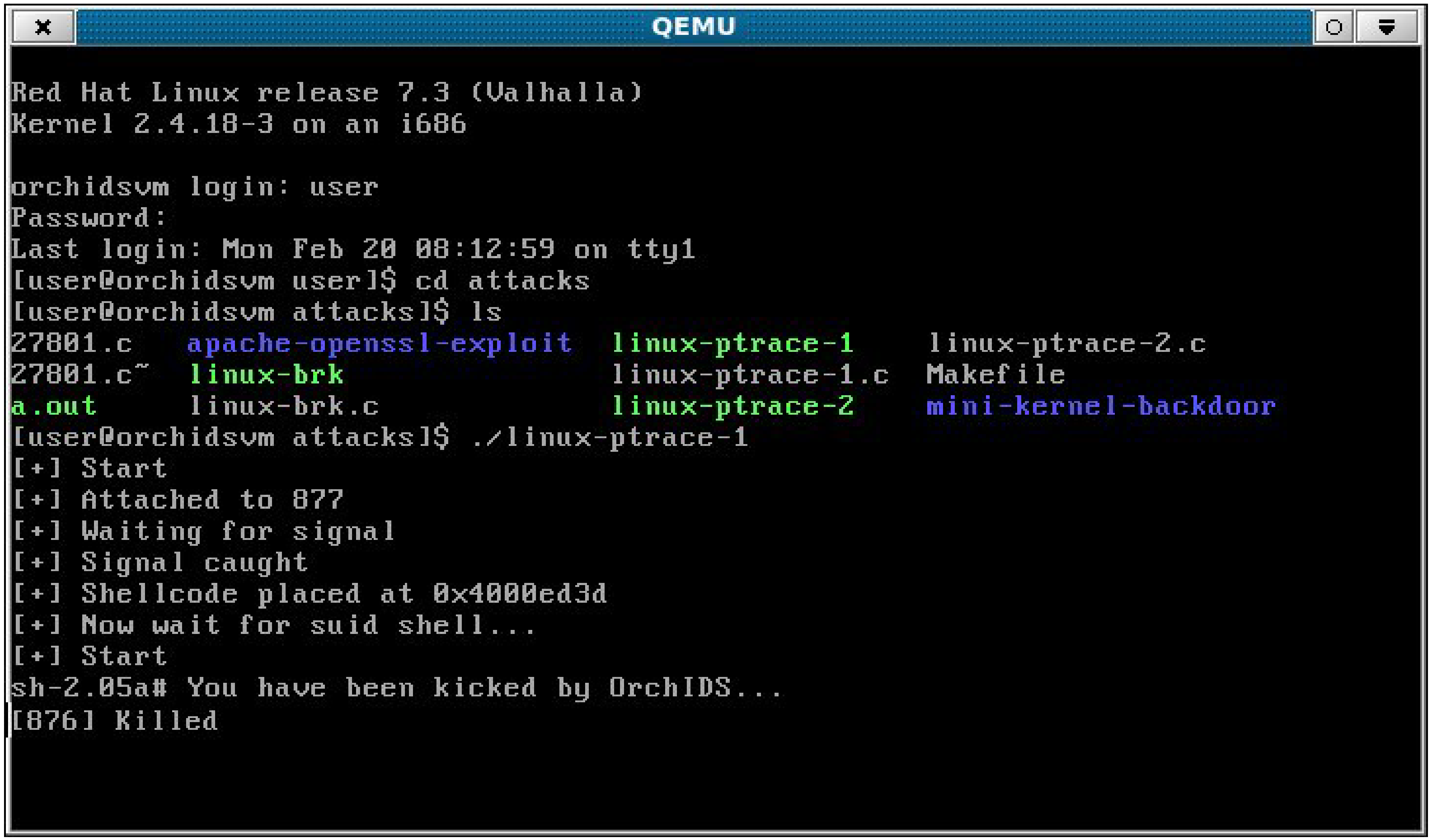
ptrace vs. ORCHIDS



Let's rerun the attack...

with ORCHIDS on, this time

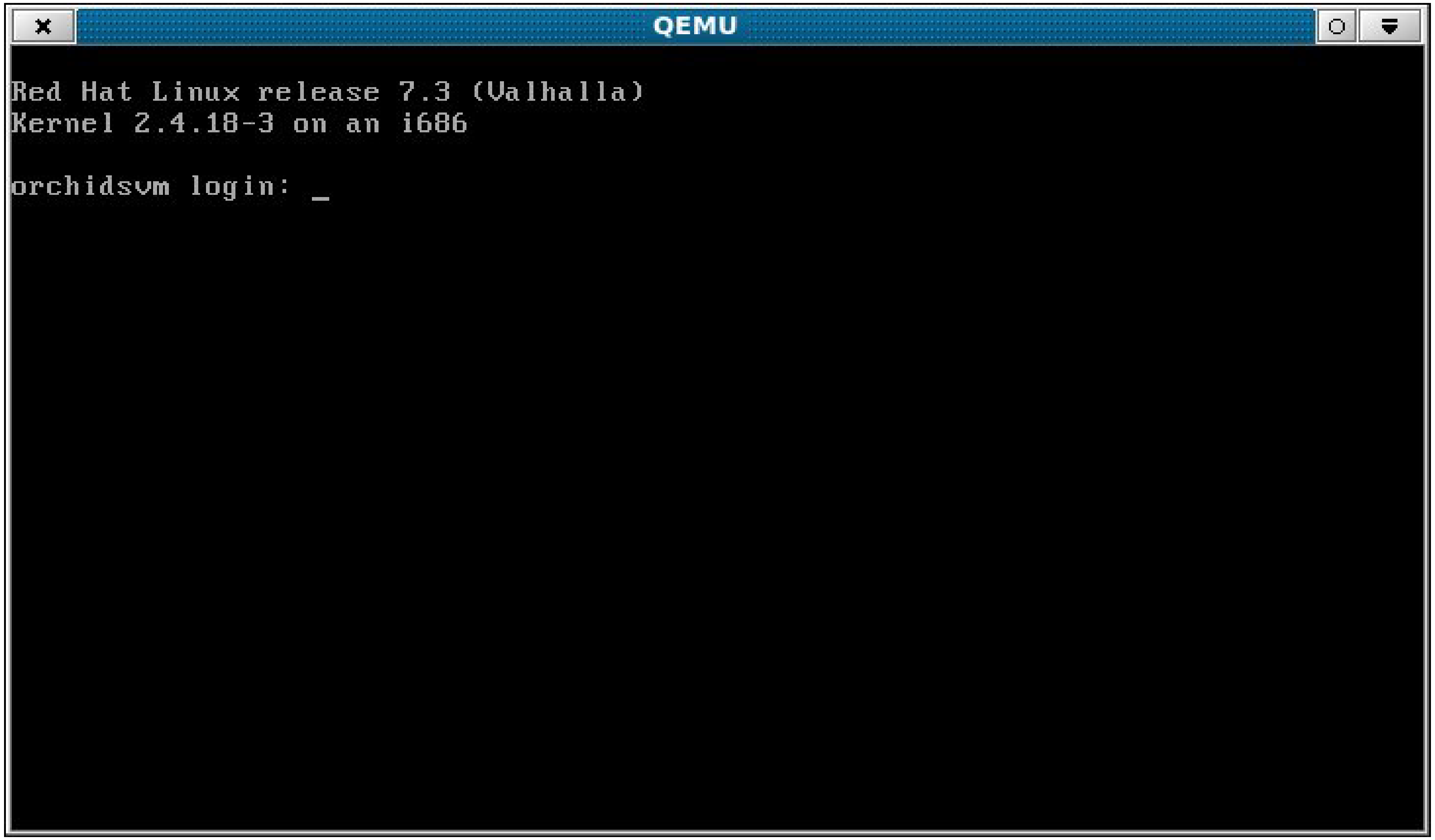
ptrace vs. ORCHIDS



The attack succeeded...

and ORCHIDS kicked the attacker out

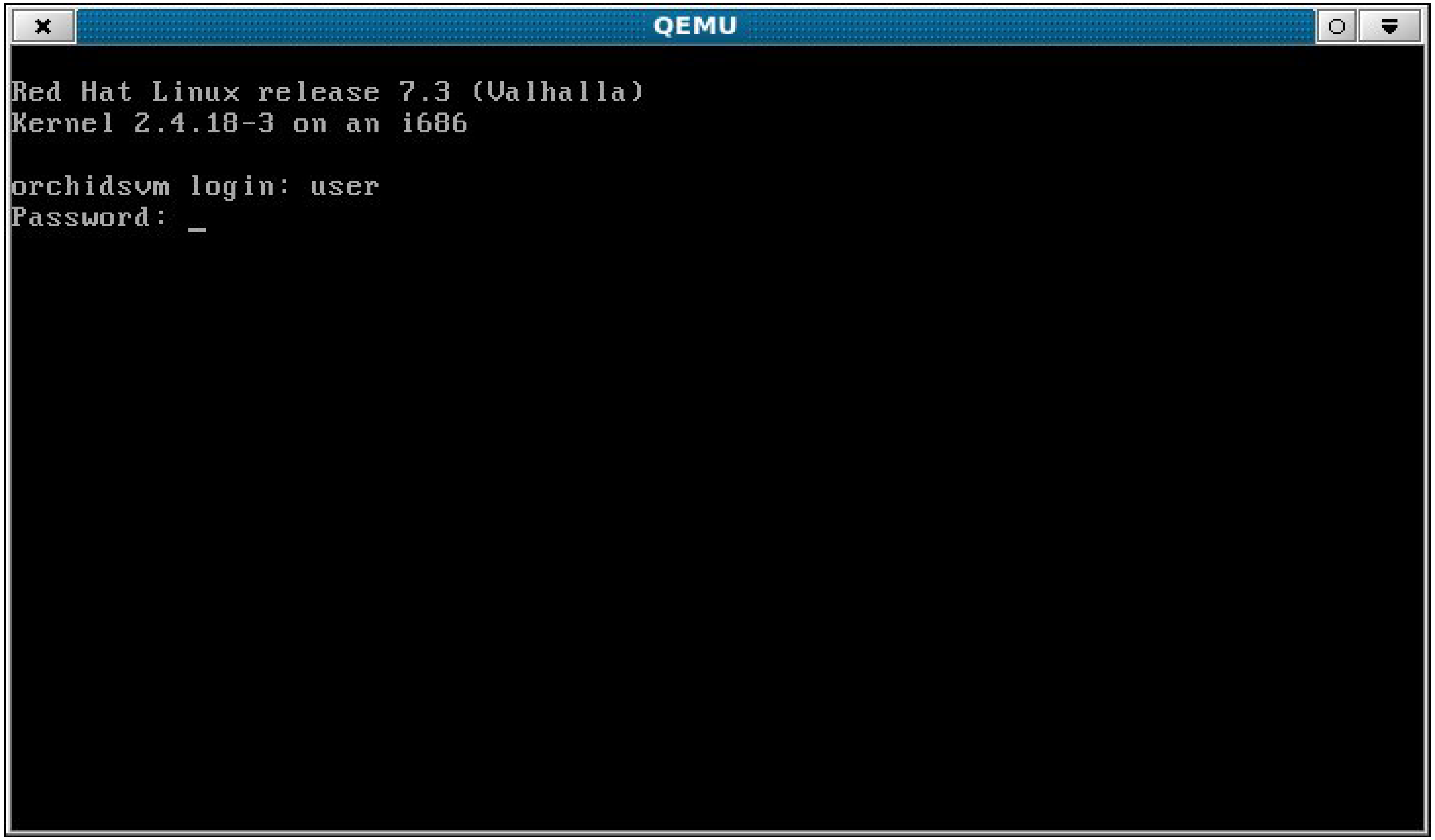
ptrace vs. ORCHIDS



The attack succeeded...

and ORCHIDS kicked the attacker out

ptrace vs. ORCHIDS

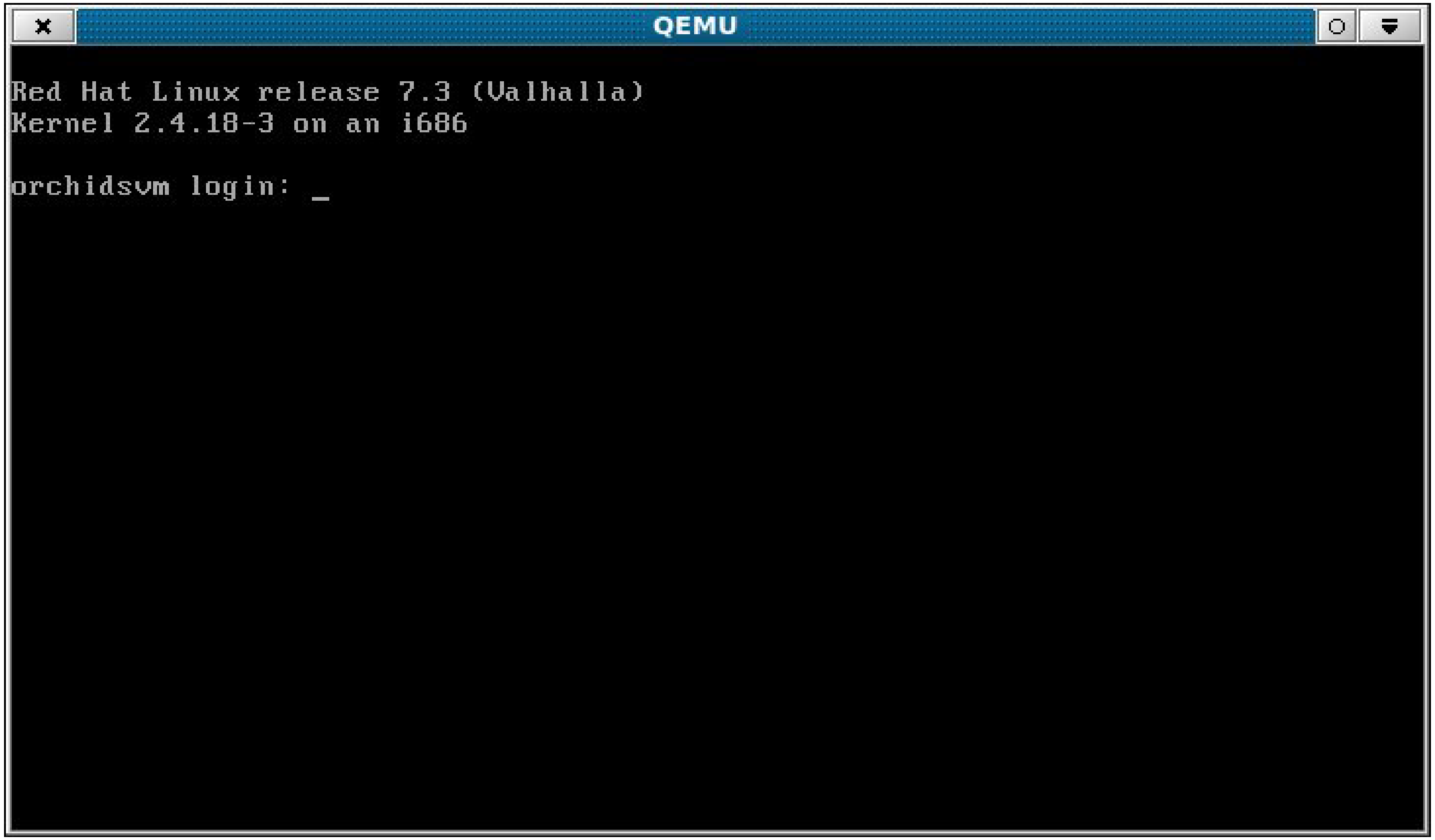


The attack succeeded...

and ORCHIDS kicked the attacker out

... and for good

ptrace vs. ORCHIDS

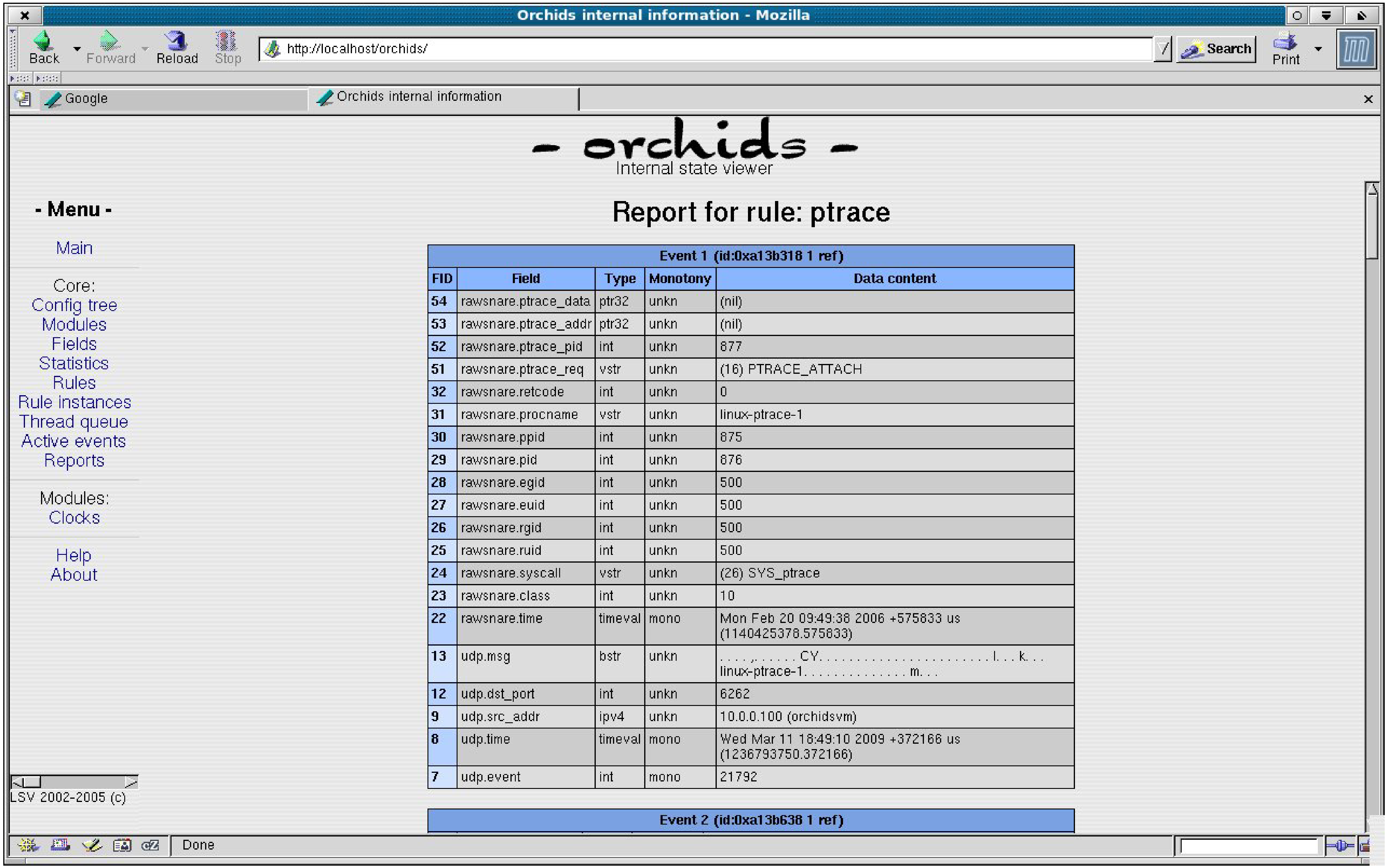


The attack succeeded...

and ORCHIDS kicked the attacker out

... and for good

Detailed reports on attacks



Report for rule: ptrace

Event 1 (id:0xa13b318 1 ref)				
FID	**Field**	**Type**	**Monotony**	**Data content**
54	rawsnare.ptrace_data	ptr32	unkn	(nil)
53	rawsnare.ptrace_addr	ptr32	unkn	(nil)
52	rawsnare.ptrace_pid	int	unkn	877
51	rawsnare.ptrace_req	vstr	unkn	(16) PTRACE_ATTACH
32	rawsnare.retcode	int	unkn	0
31	rawsnare.procname	vstr	unkn	linux-ptrace-1
30	rawsnare.ppid	int	unkn	875
29	rawsnare.pid	int	unkn	876
28	rawsnare.egid	int	unkn	500
27	rawsnare.euid	int	unkn	500
26	rawsnare.rgid	int	unkn	500
25	rawsnare.ruid	int	unkn	500
24	rawsnare.syscall	vstr	unkn	(26) SYS_ptrace
23	rawsnare.class	int	unkn	10
22	rawsnare.time	timeval	mono	Mon Feb 20 09:49:38 2006 +575833 us (1140425378.575833)
13	udp.msg	bstr	unkn	. . . . ,. . . . . . CY. . . . . l. . . k. . . linux-ptrace-1. . . . . m. . .
12	udp.dst_port	int	unkn	6262
9	udp.src_addr	ipv4	unkn	10.0.0.100 (orchidsvm)
8	udp.time	timeval	mono	Wed Mar 11 18:49:10 2009 +372166 us (1236793750.372166)
7	udp.event	int	mono	21792

Event 2 (id:0xa13b638 1 ref)

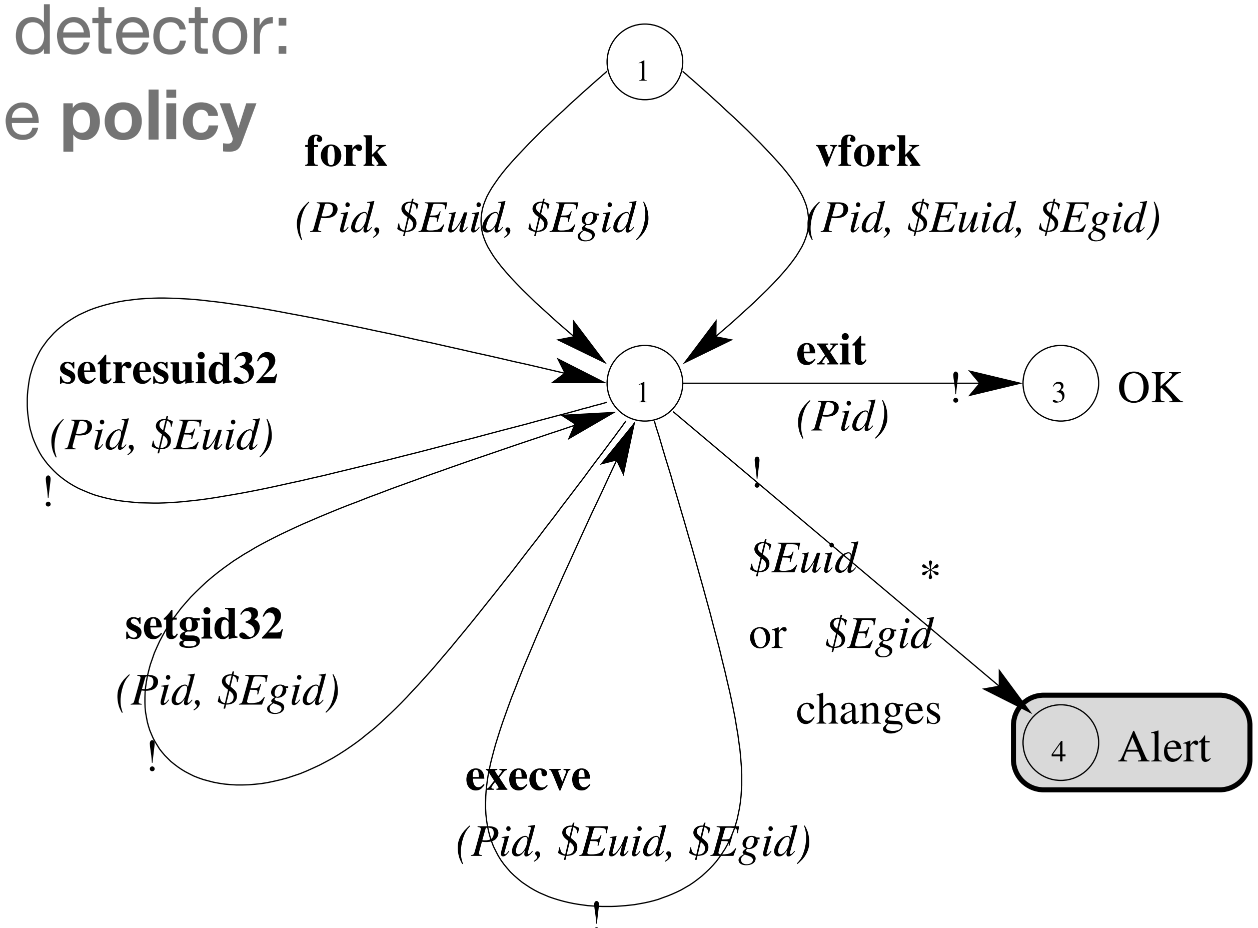
Time for a demo, for real

- The **semtex** local-to-root exploit (sd@fucksheep.org, May 2013)
  Bug:
  In file `kernel/events/core.c`:   `int event_id = event->attr_config; /* u64 */`

- Caught by the **pid_tracker** OrchIDS rule,

  an (almost) universal local-to-root exploit detector:
  checks **conformance** to Linux uid change **policy**

- The same rule catches:
  do_brk (2003)
  do_mremap (2004)
  do_mmap (2005)
  vmsplice (2008)



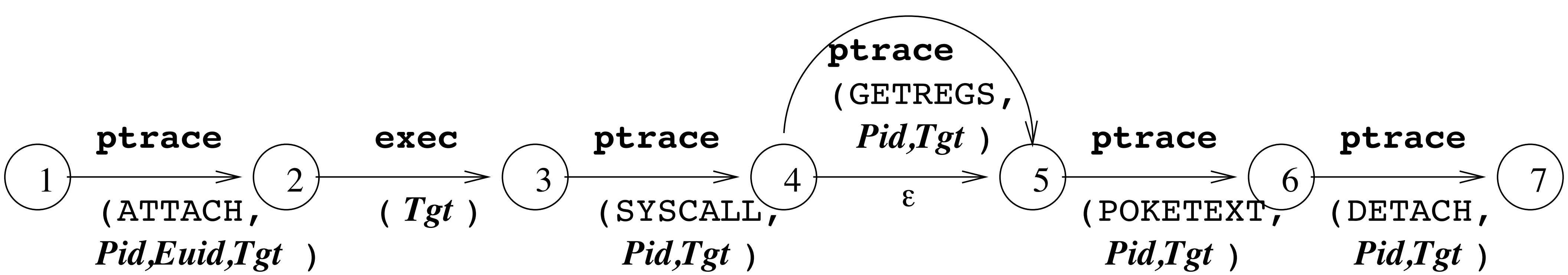
How it works

- The monitored machines collect **events:**

```
open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)
```

...

- We look for **signatures** that identify the attack:



How it works

- The monitored machines collect **events:**

```
Jan 26 20:34:13 darkstar kernel: PPP line discipline registered.
Jan 26 20:34:13 darkstar kernel: cs: IO port probe 0x0100-0x03ff: excluding 0x100-0x107
Jan 26 20:34:13 darkstar kernel: cs: IO port probe 0x0a20-0x0a27: clean.
Jan 26 20:34:13 darkstar kernel: cs: memory probe 0x0c0000-0x0fffff: excluding 0xe0000-0xfffff
Jan 26 20:34:13 darkstar kernel: tty01 at 0x02f8 (irq = 3) is a 16550A
Jan 26 20:34:49 darkstar login[87]: ROOT LOGIN on `tty1'
Jan 26 20:42:03 darkstar init: Switching to runlevel: 0
Jan 26 22:27:00 darkstar syslogd 1.3-0#: restart.
Jan 26 22:27:01 darkstar kernel: Loaded 4342 symbols from /boot/System.map.
Jan 26 22:27:01 darkstar kernel: Symbols match kernel version.
Jan 26 22:37:04 darkstar auditd[88]: open("/etc/passwd","r")=4
Jan 26 22:37:04 darkstar kernel: NET3: Unix domain sockets 0.13 for Linux NET3.035.
Jan 26 22:37:04 darkstar kernel: VFS: Diskquotas version dquot_5.6.0 initialized
Jan 26 22:37:04 darkstar auditd[88]: read(4,1024)=573
Jan 26 20:37:04 darkstar auditd[88]: read(4,1024)=-1
Jan 26 20:37:04 darkstar auditd[89]: ptrace(PTRACE_ATTACH,88)=0
Jan 26 20:37:04 darkstar auditd[88]: close(4)=0
...
```

- We look for **signatures** that identify the attack:

```
rule ptrace
{
  state init
  {
    if (.rawsnare.syscall == "(26) SYS_ptrace" &&
        .rawsnare.ptrace_req == "(16) PTRACE_ATTACH" &&
        .rawsnare.euid != 0 &&
        .rawsnare.egid != 0)
      goto ptrace_attach;
  }
```

```
state ptrace_attach
{
  $attack_pid = .rawsnare.pid;
  $target_pid = .rawsnare.ptrace_pid;
  $attacker_uid = .rawsnare.euid;
  $counter = 0;

  if (.rawsnare.syscall == "(11) SYS_execve" &&
      .rawsnare.path == "/sbin/modprobe" &&
      .rawsnare.pid == $target_pid)
    goto exec_modprobe;
}
...
```

How it works

Flow of events:

```
open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)
```

Orchids threads: (none)

How it works

Flow of events:

```
open (”/etc/passwd”, ”r”, pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog=”modprobe”, pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)
```

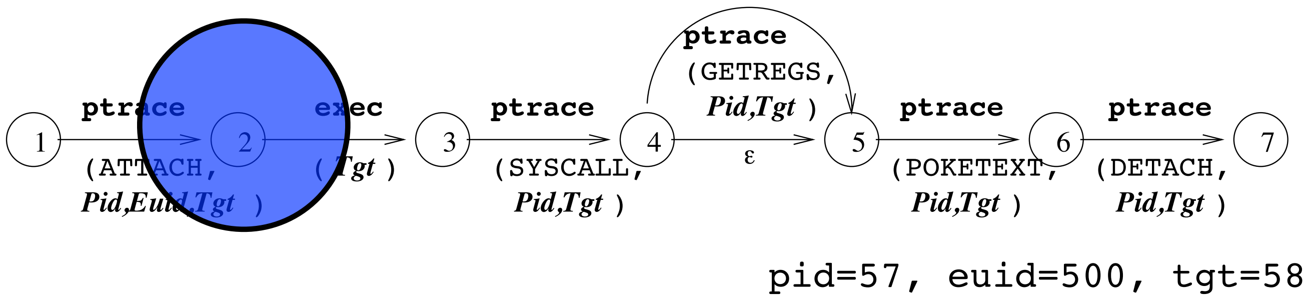
Orchids threads: (none)

How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)	ptrace (SYSCALL, pid=100, 101)
ptrace (ATTACH, pid=57, euid=500, 58)	ptrace (GETREGS, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exec (prog="modprobe", pid=101)	ptrace (POKETEXT, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exit (pid=58)	ptrace (DETACH, pid=100, 101)

Orchids threads:



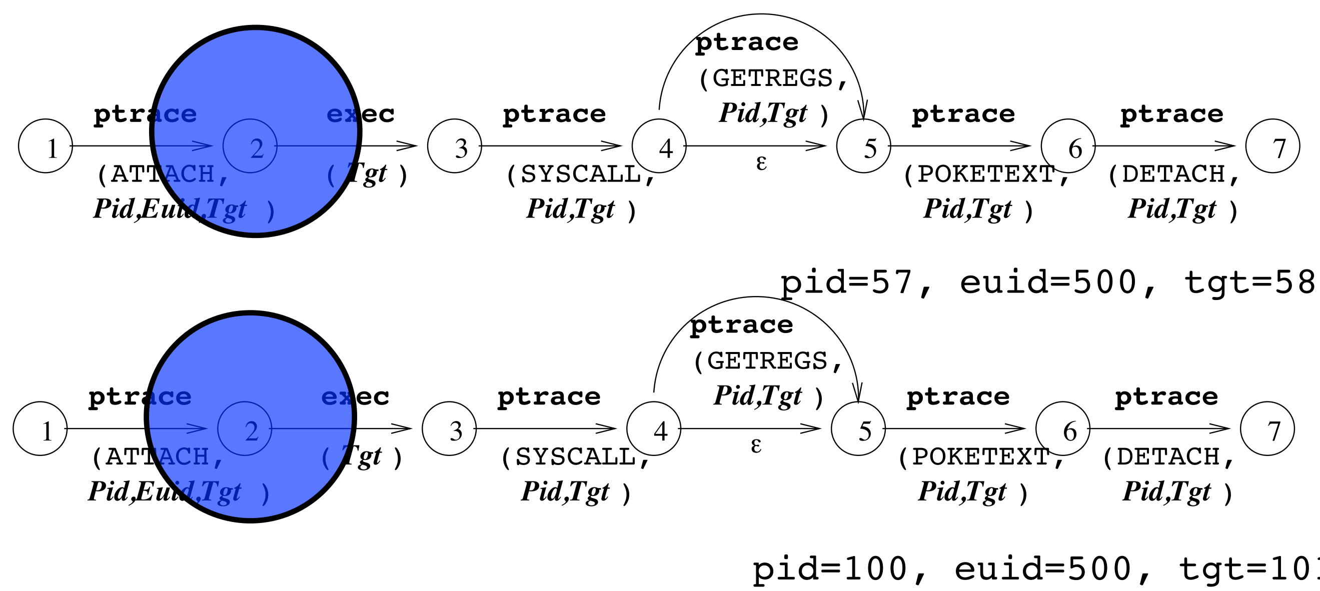
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



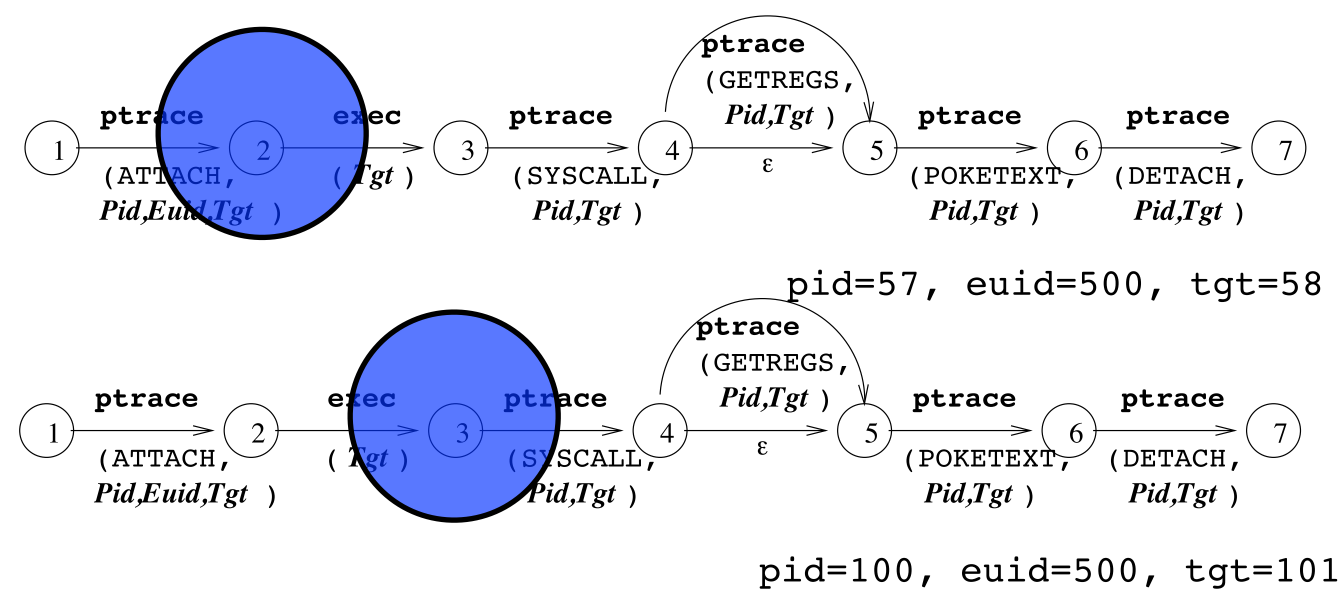
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



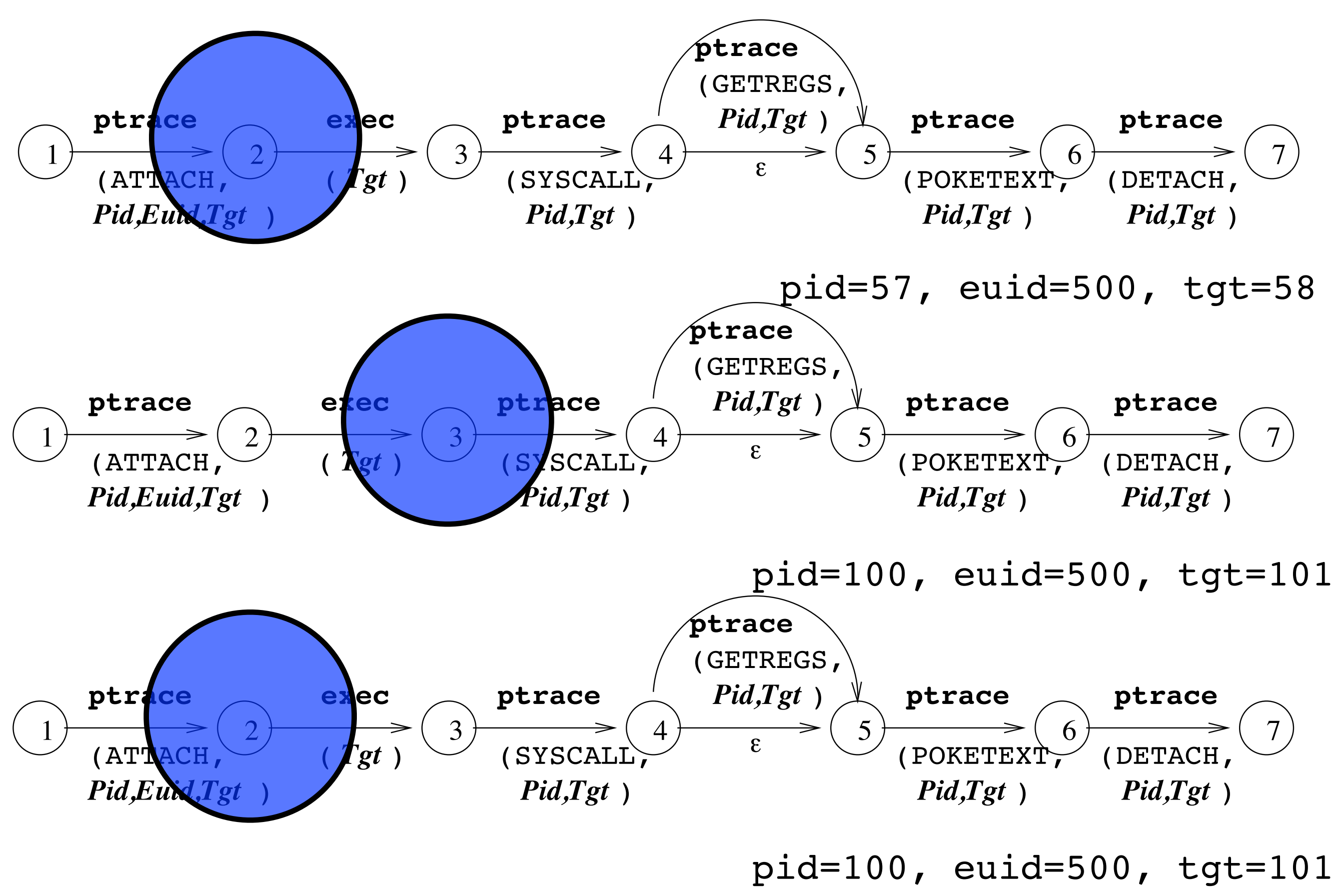
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:

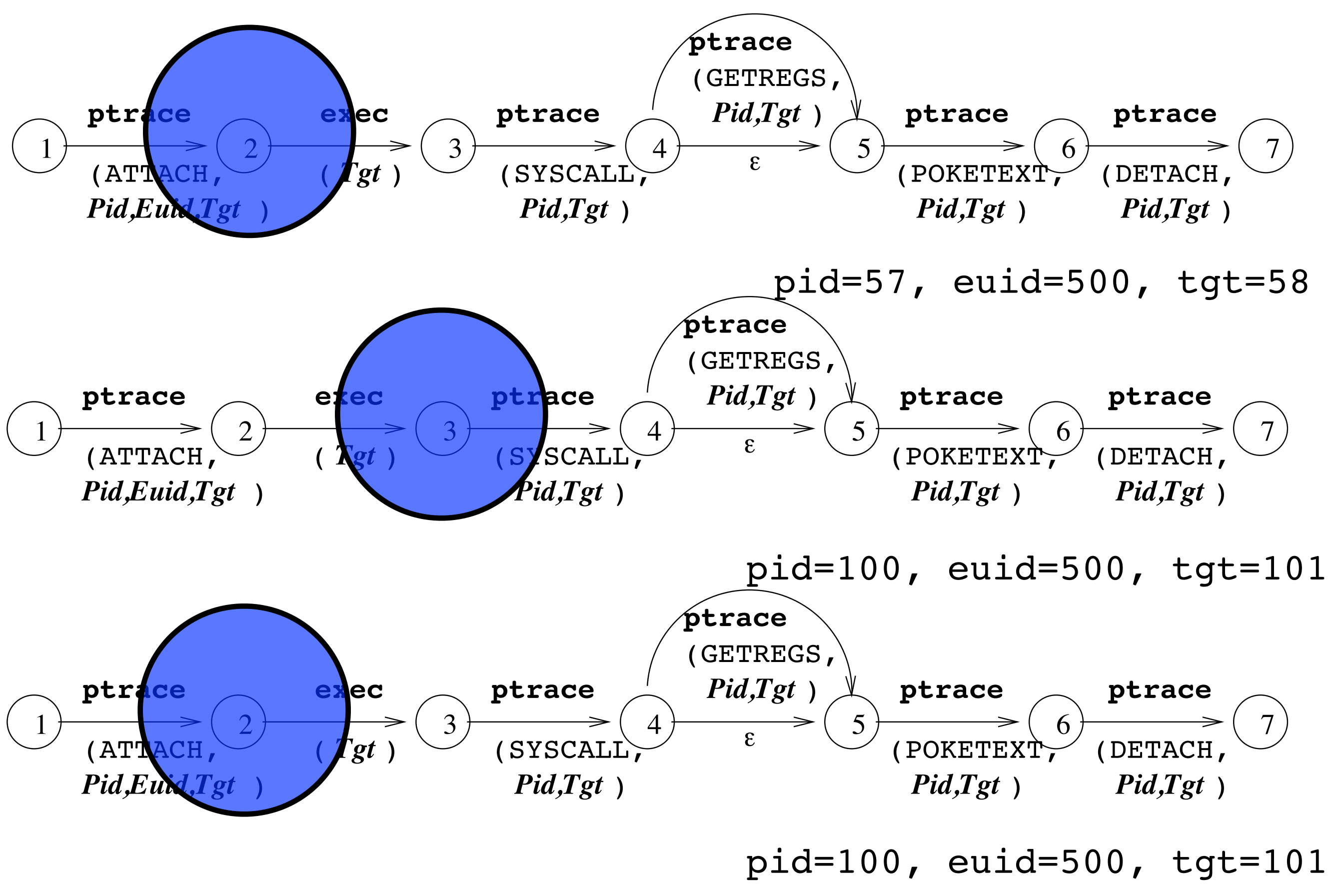


How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



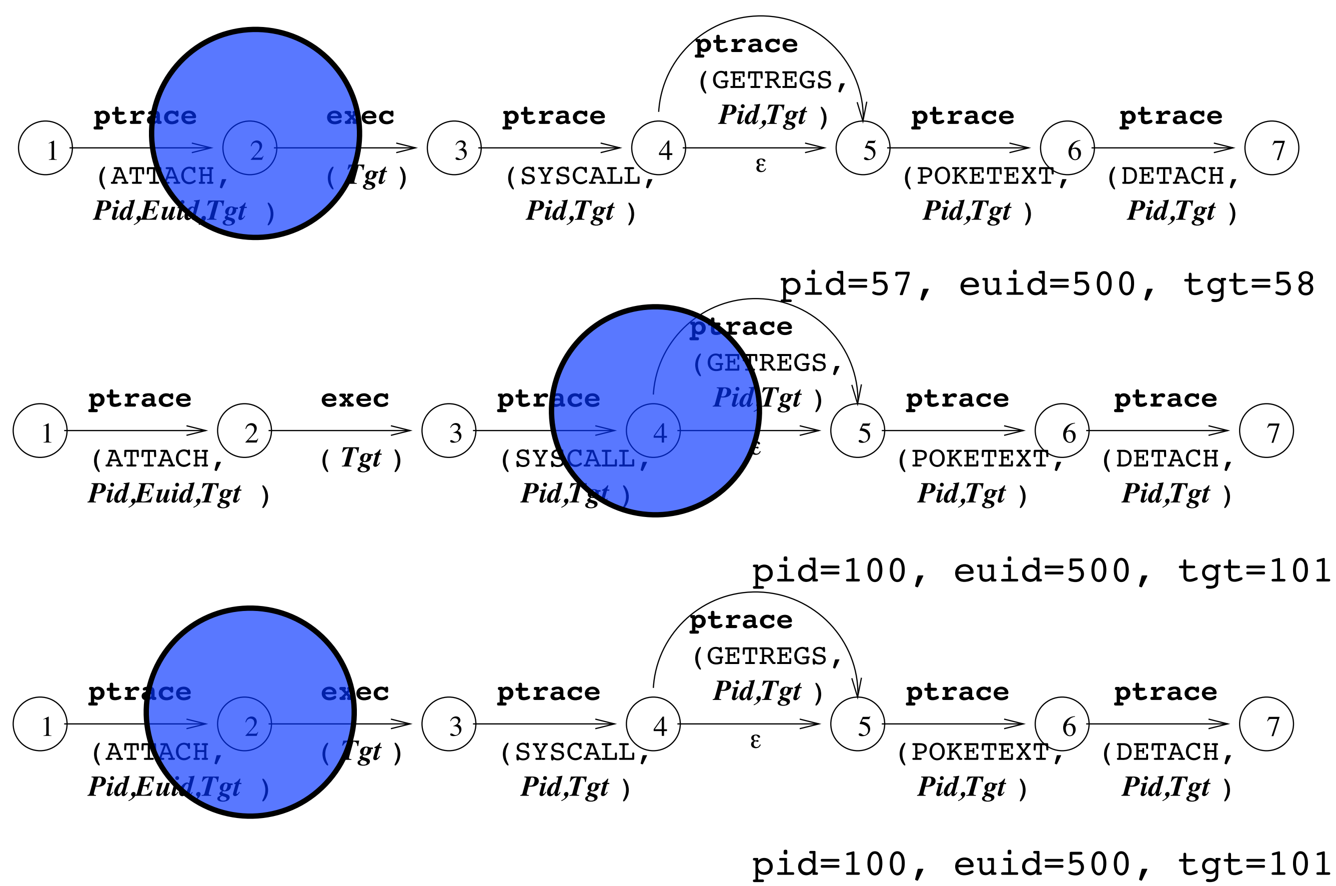
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



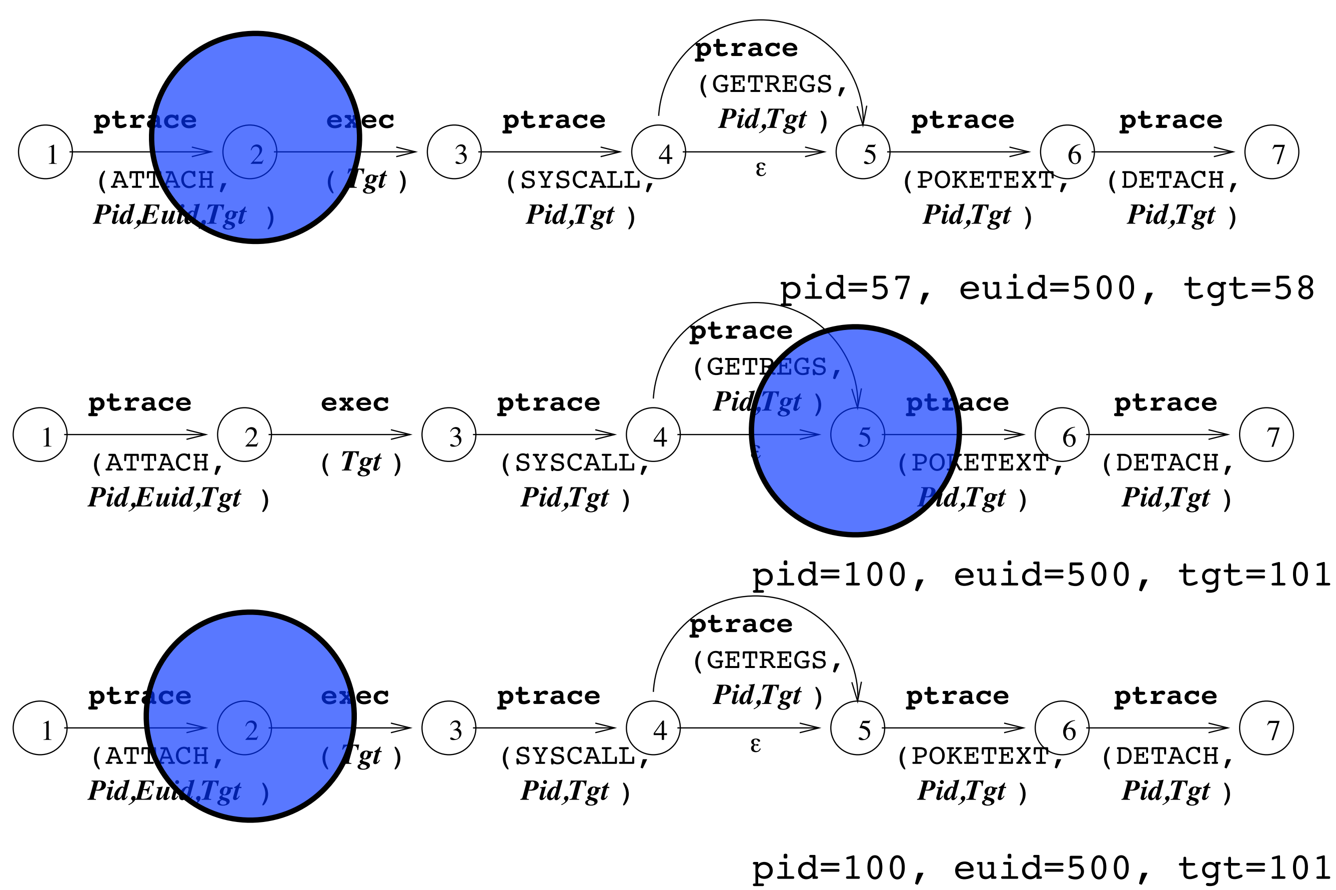
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



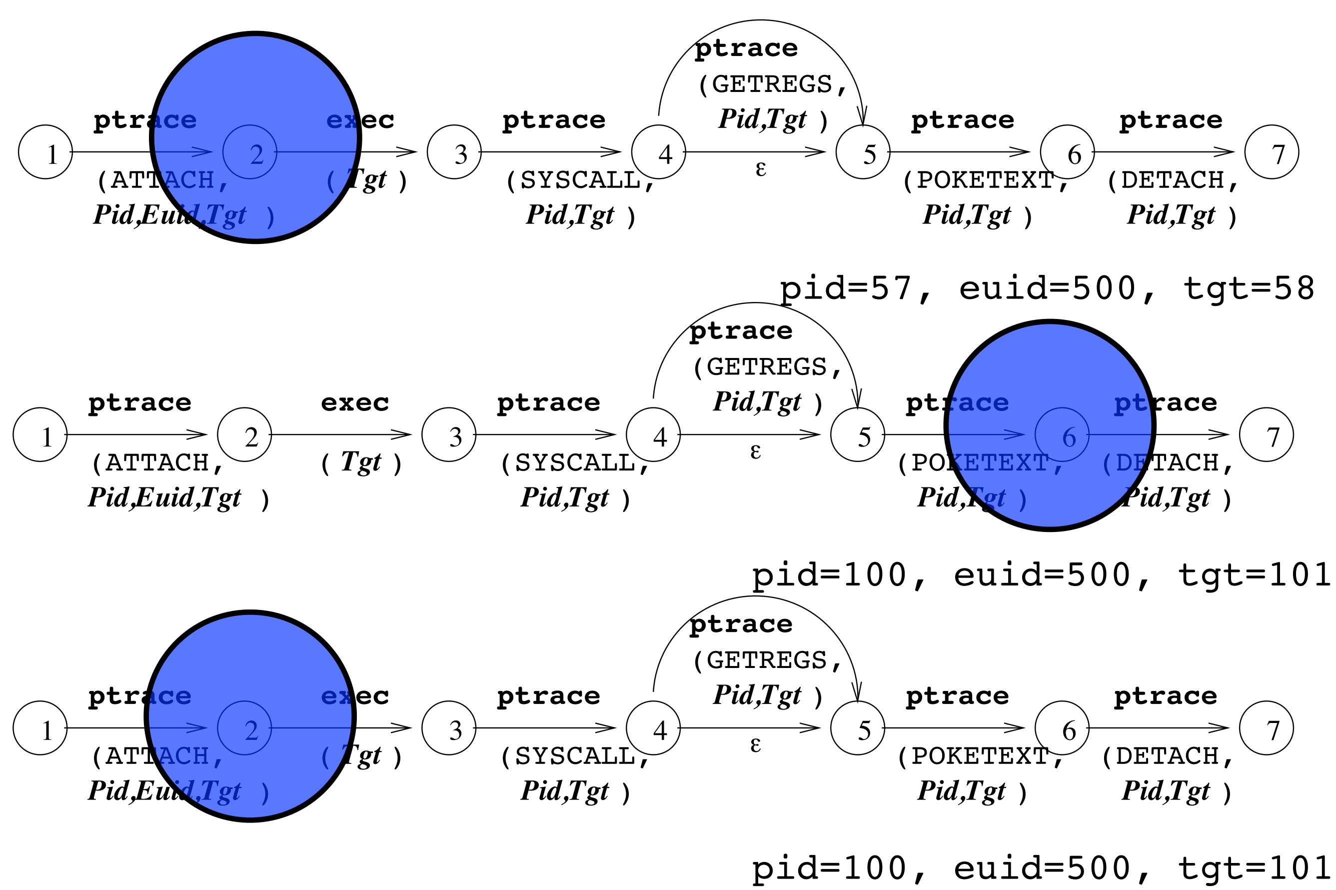
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



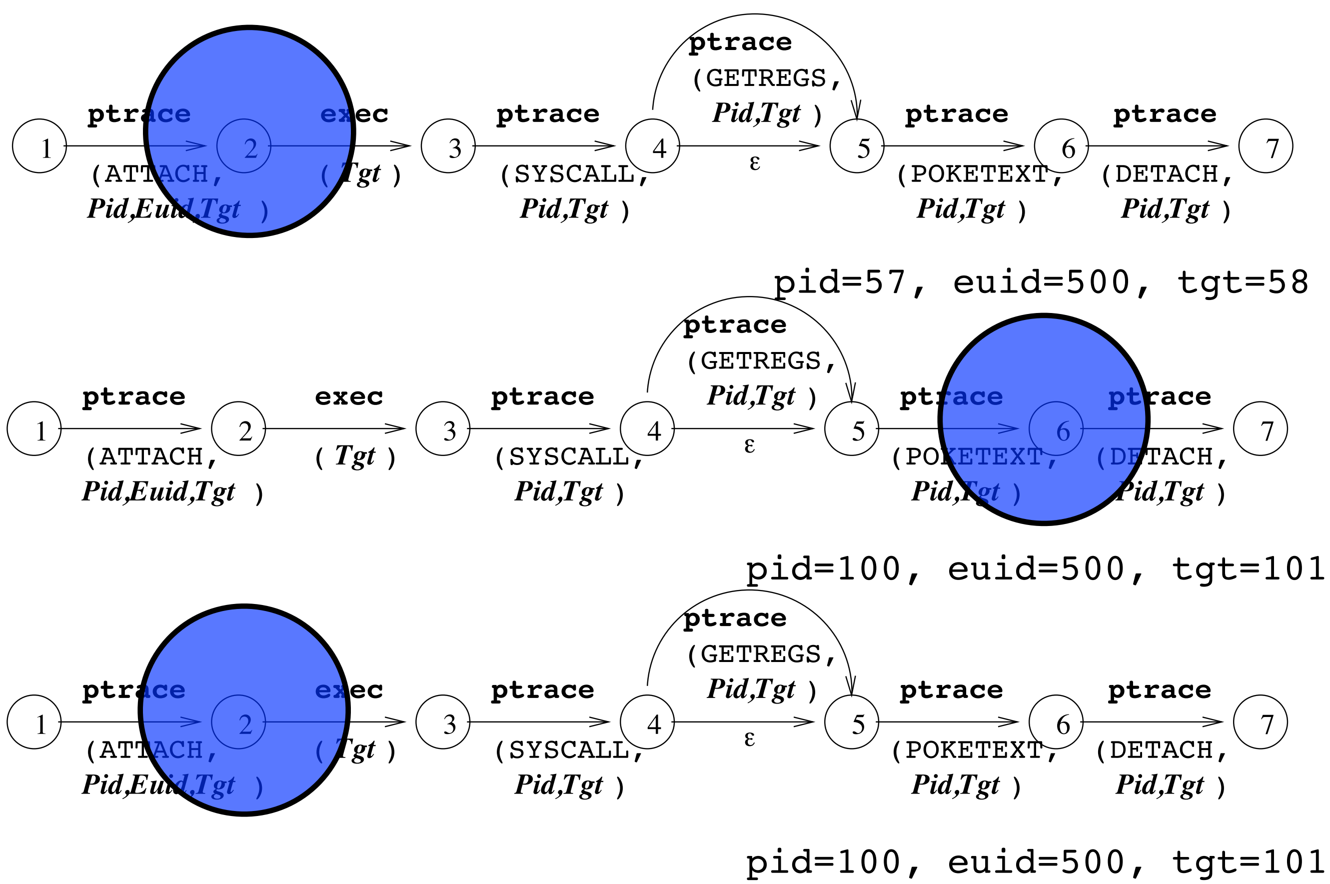
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



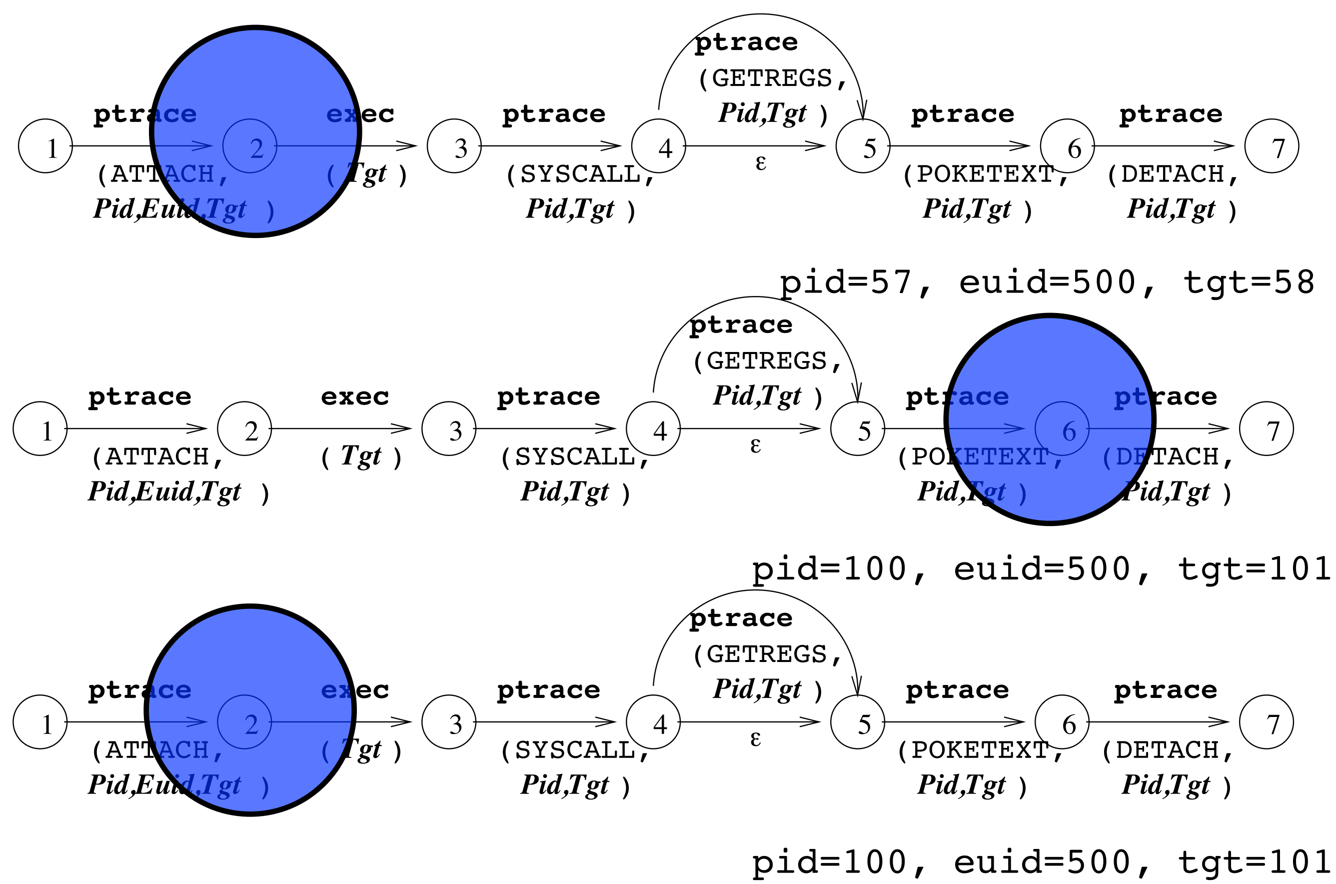
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



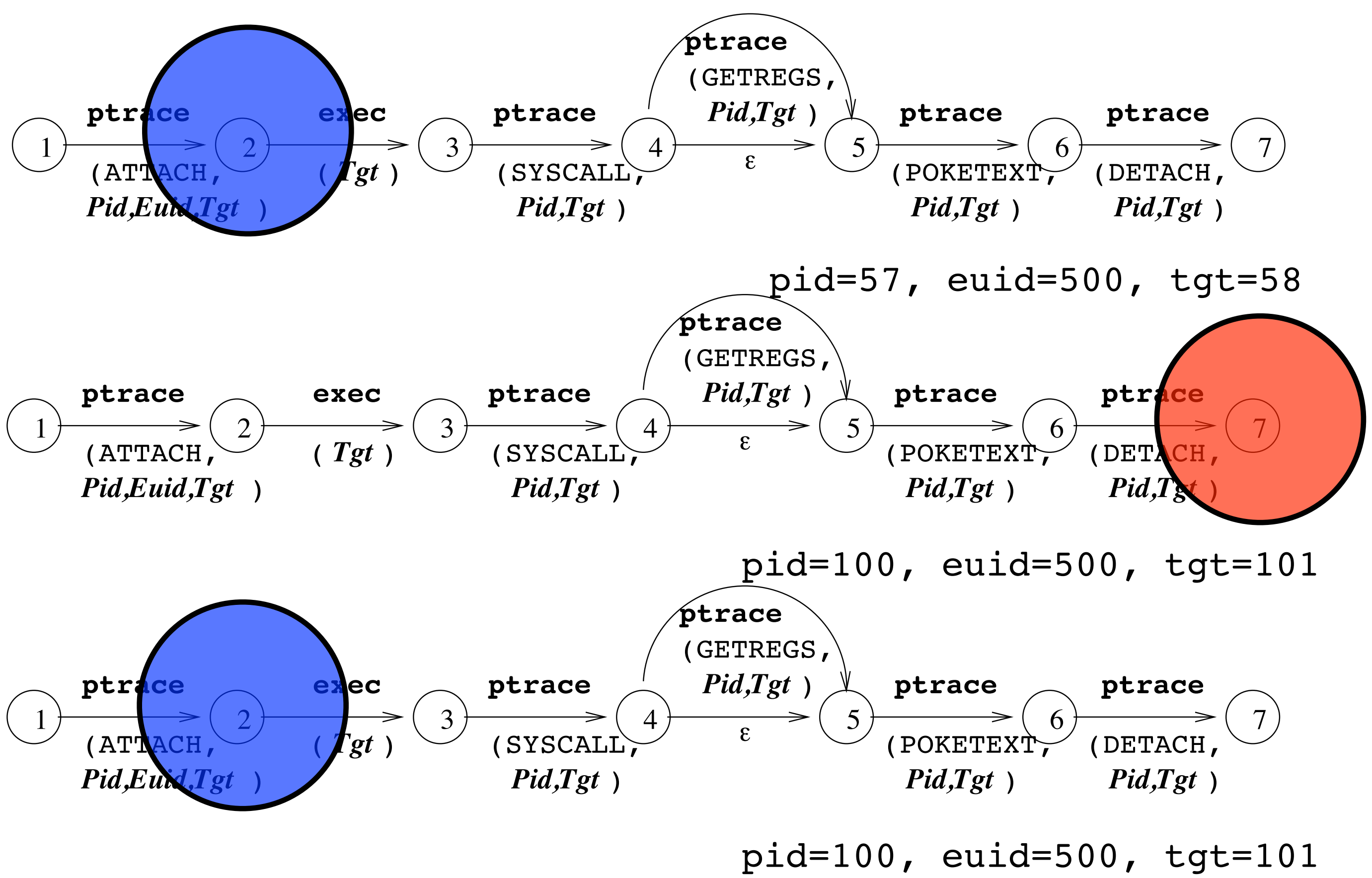
How it works

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

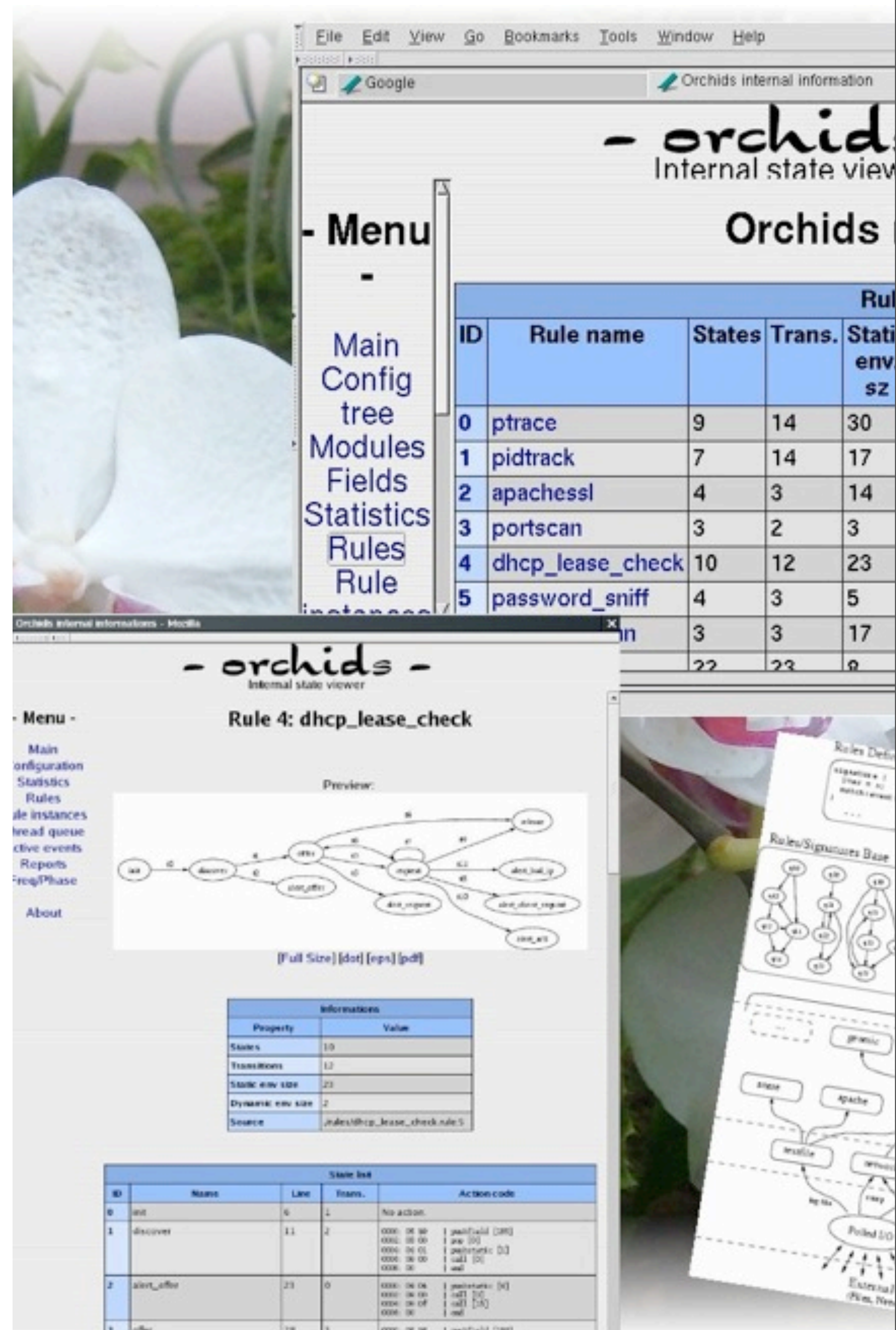
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

Orchids threads:



Related work

- **P-Best** [Lindqvist-Porras 1999]
- **Statl** [Eckmann-Vigna-Kemmerer 2000]
- **Chronicles** [e.g., Morin-Debar 2003]
- **Lambda** [Cuppens-Miege 2002]
- **Sutekh** [Pouzol-Ducassé 2002]
- **Blare** [George-VietTriemTong-Mé 2009]
- **RV-Monitor** [Rosu et al. 2008, 09, 12, 14]
- ... and probably many others



Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

Semantics, and detection algorithms

- Semantics: what should Orchids detect?
- Algorithm: how should I detect it?
  (This is what I showed you.)
- Semantics **dictates** the algorithm.
- ... somehow opposite to the average coding attitude
  - we like to think algorithmically
  - we are eager to **code**

http://www.sadgrin.com/wp-content/uploads/2013/03/geek-300x300.jpg

Semantics, and detection algorithms

- Semantics: what should Orchids detect?
- Algorithm: how should I detect it?
  (This is what I showed you.)
- Semantics **dictates** the algorithm.
- ... somehow opposite to the average coding attitude
  - we like to think algorithmically
  - we are eager to **code**

http://www.sadgrin.com/wp-content/uploads/2013/03/geek-300x300.jpg

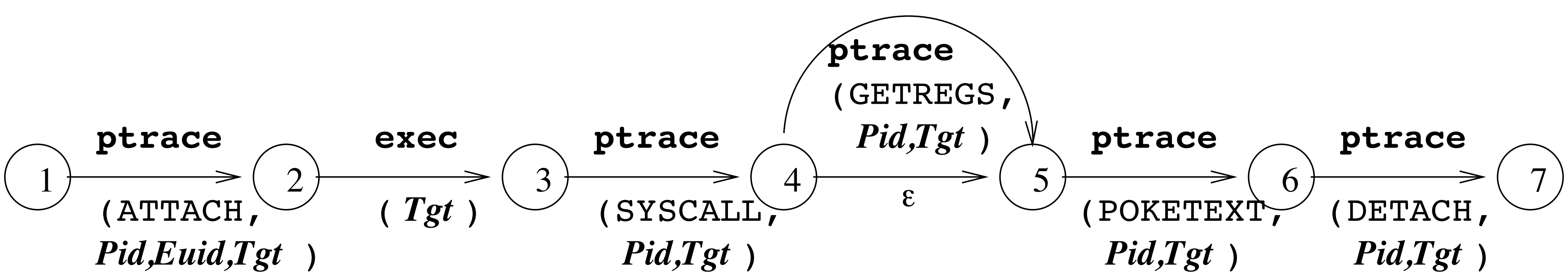
Semantics, 1

- ORCHIDS looks for subsequences of events («**runs**»)

A
`ptrace`(`ATTACH`, ...)
B
A
`exec`(...)
`ptrace`(`SYSCALL`, ...)
A

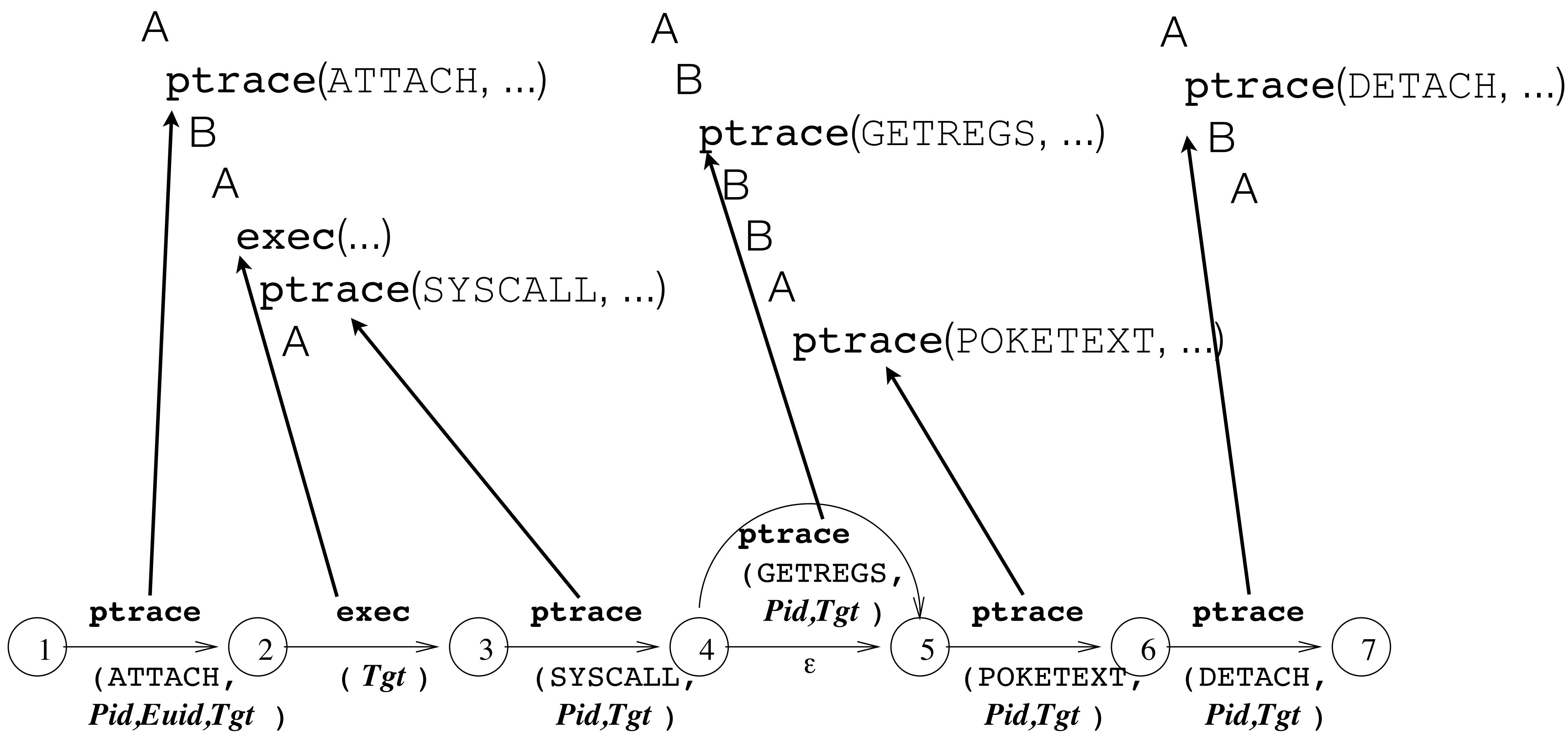
A
B
`ptrace`(`GETREGS`, ...)
B
B
A
`ptrace`(`POKETEXT`, ...)

A
`ptrace`(`DETACH`, ...)
B
A



Semantics, 1

- ORCHIDS looks for subsequences of events («**runs**»)

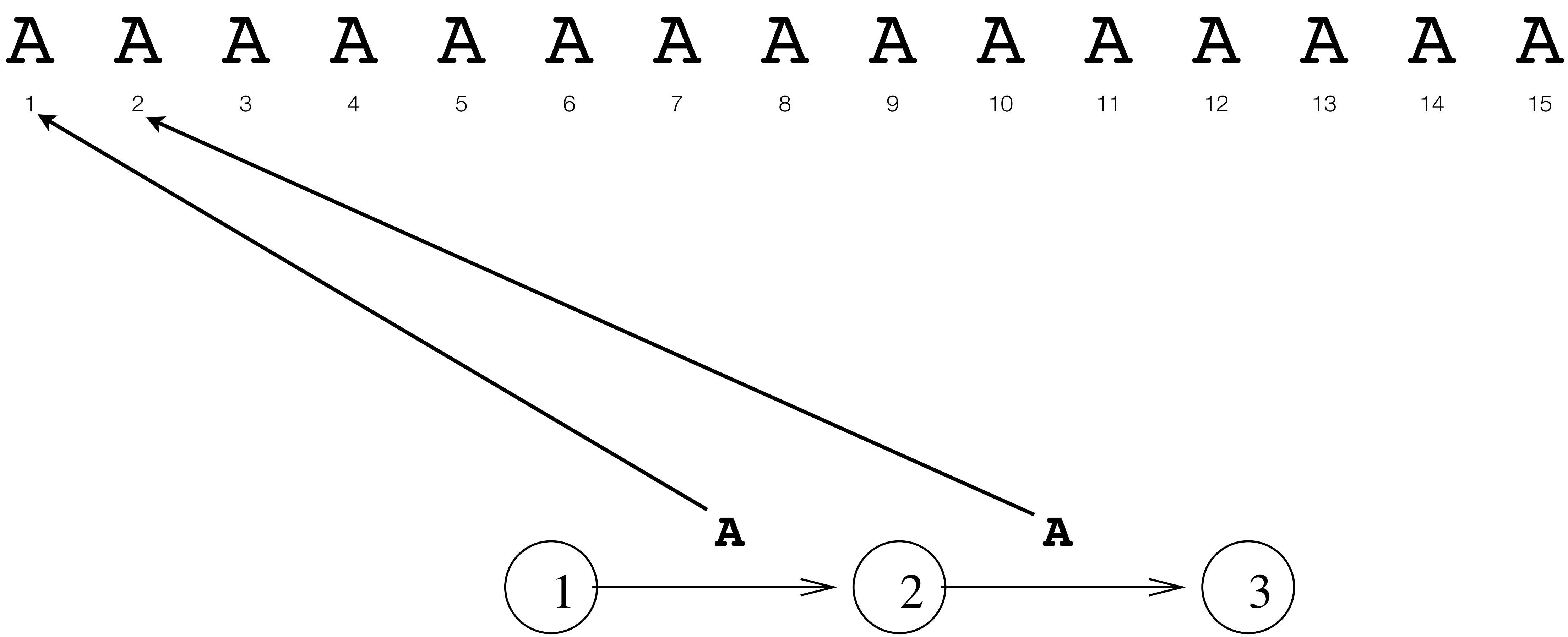


Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events

- In this (simple) example, **many** possible runs (even by fixing the start event)

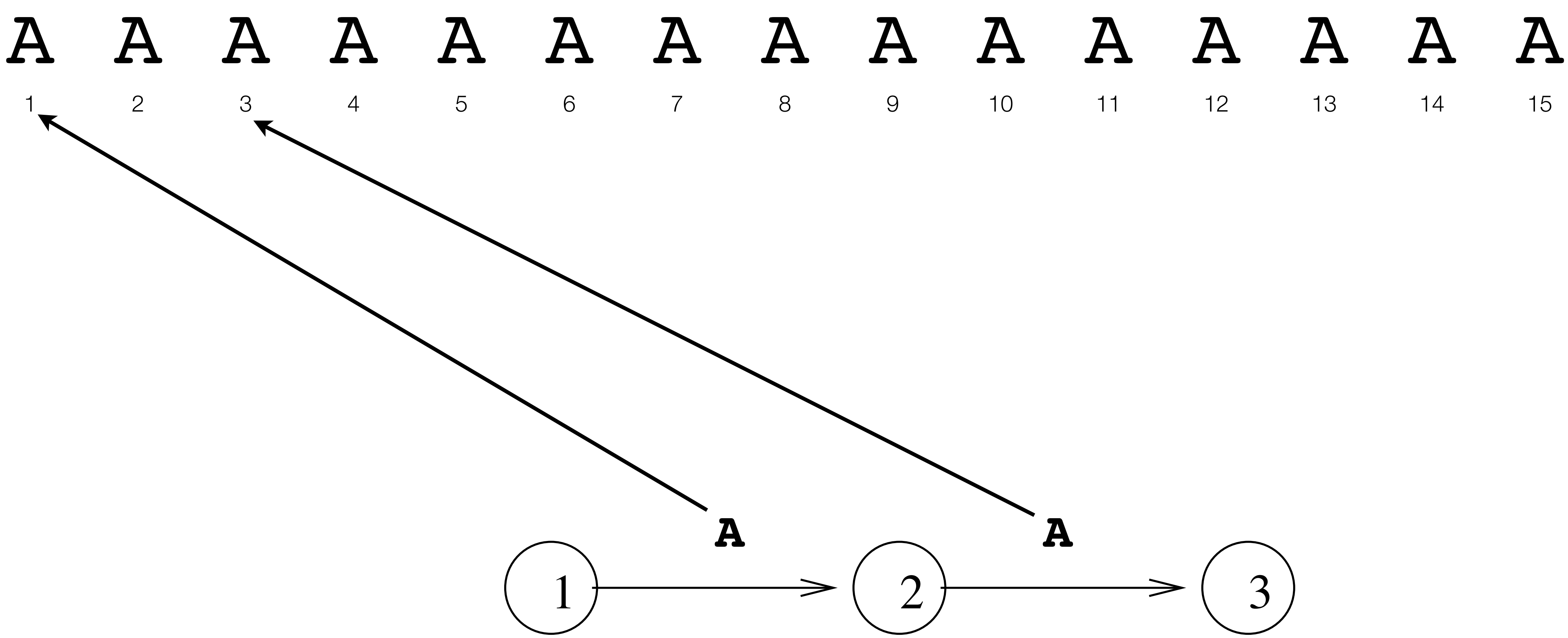
Here is one:



Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events
- In this (simple) example, **many** possible runs (even by fixing the start event)

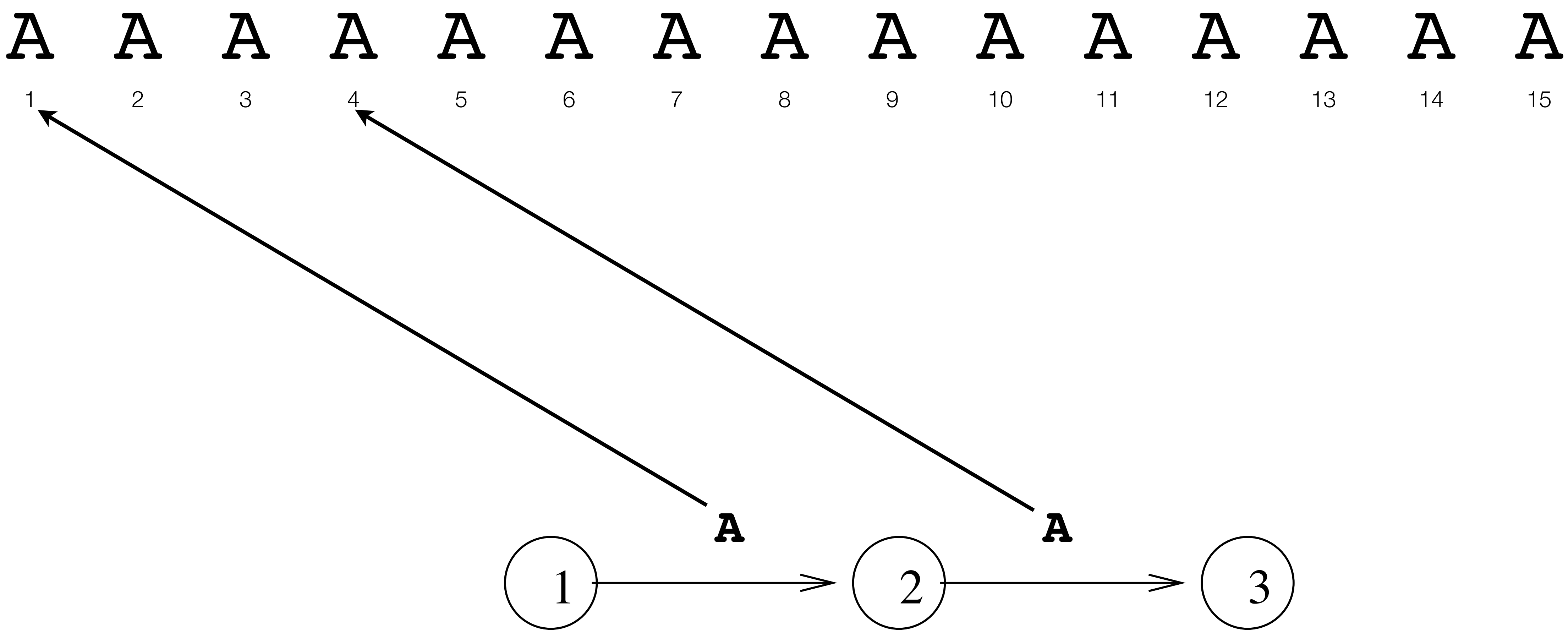
Another one:



Semantics, 2: «shortest runs»

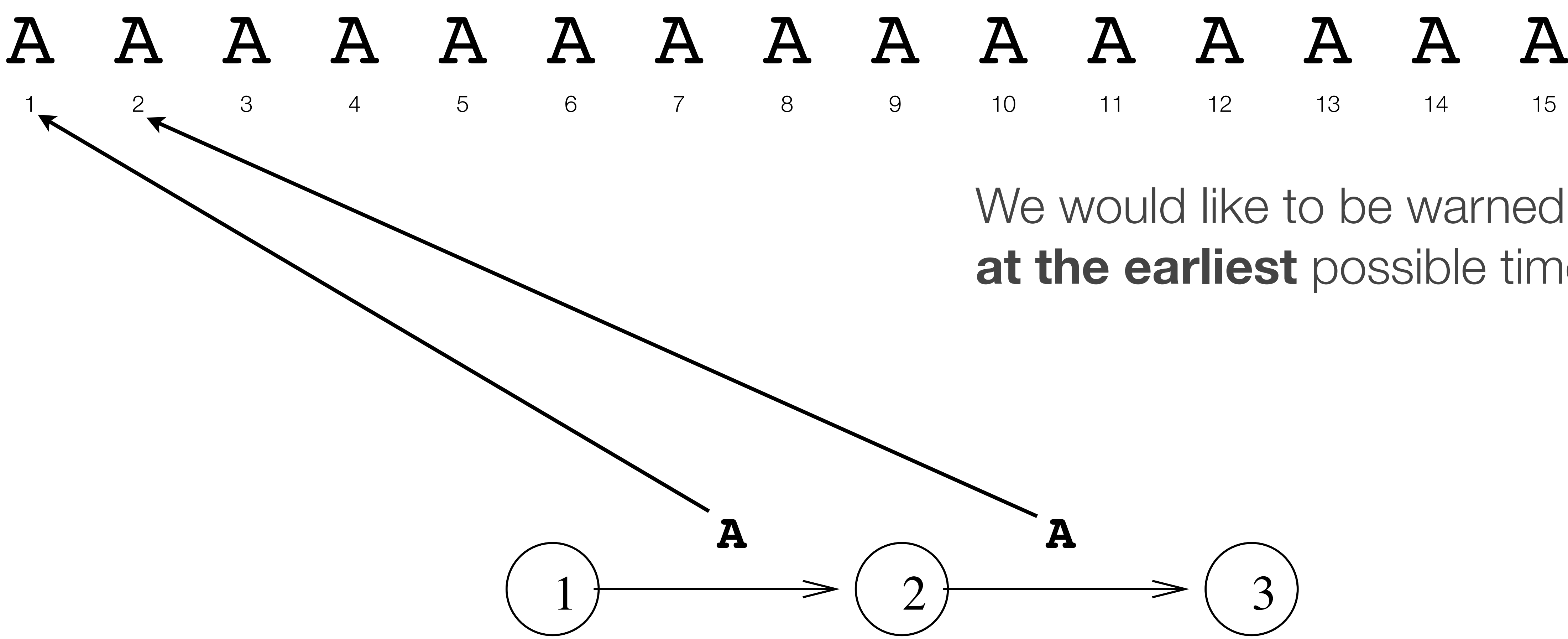
- ORCHIDS looks for subsequences of events
- In this (simple) example, **many** possible runs (even by fixing the start event)

Yet another:



Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events
- In this (simple) example, **many** possible runs (even by fixing the start event)

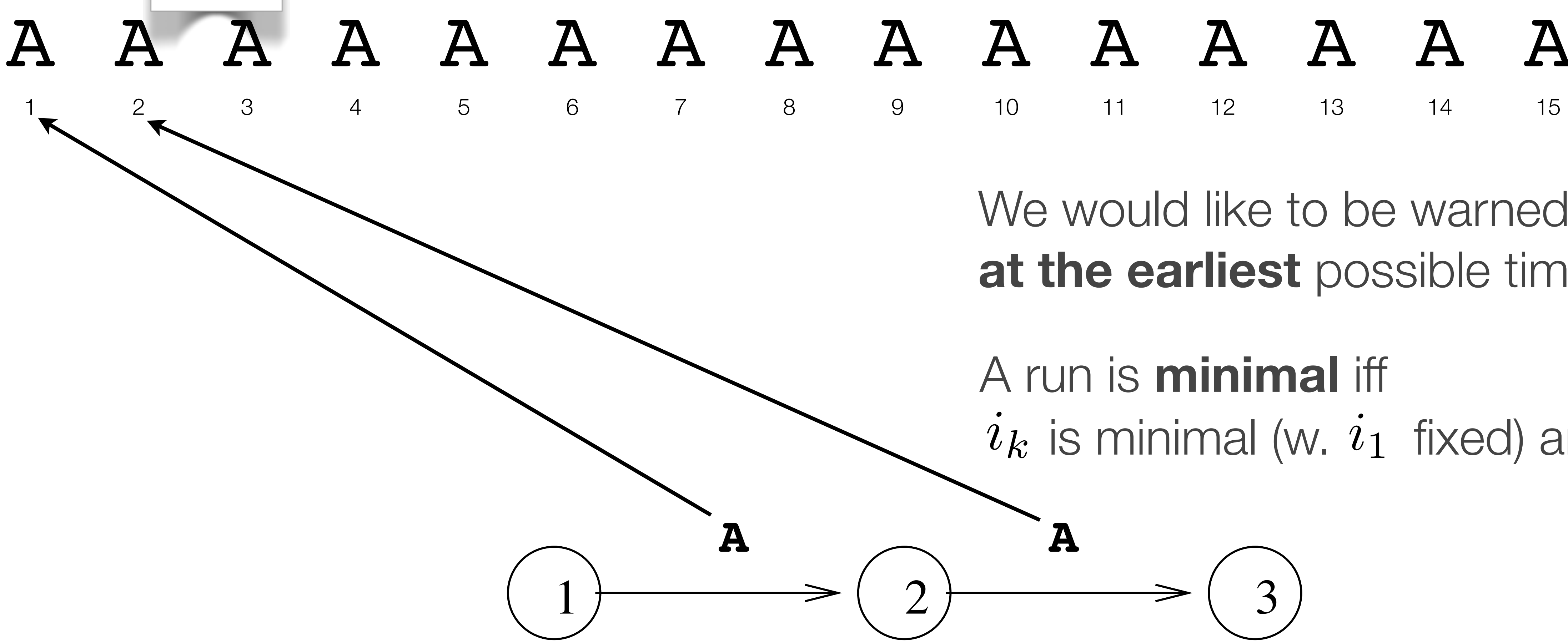


We would like to be warned **at the earliest** possible time

Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
(Here, 1 2 .)



We would like to be warned
at the earliest possible time

A run is **minimal** iff
i_k is minimal (w. i_1 fixed) and ...

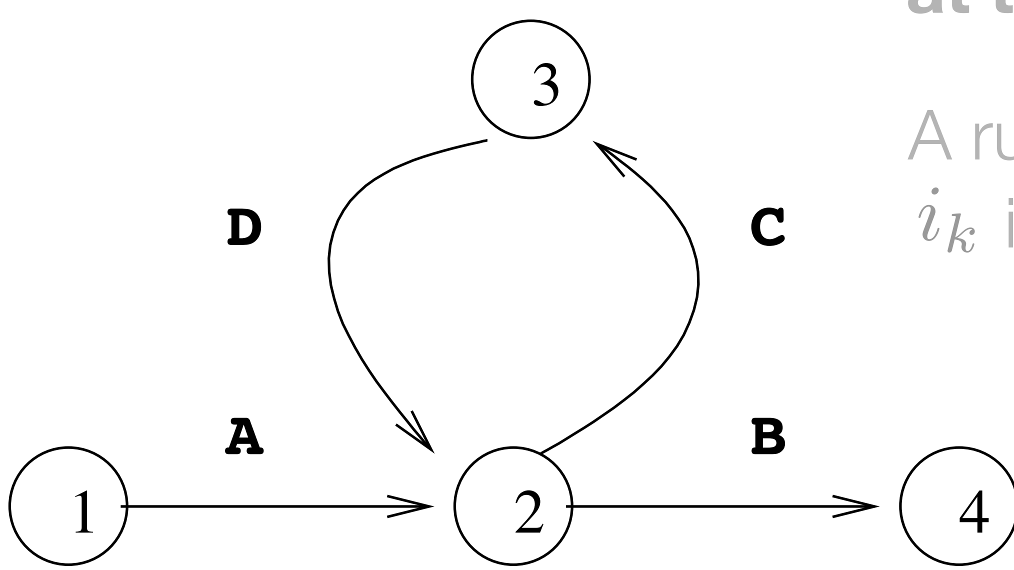
Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  Another example:

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑	↑	↑					↑							

1 2 3 8



We would like to be warned **at the earliest** possible time

A run is **minimal** iff
i_k is minimal (w. i_1 fixed) and ...

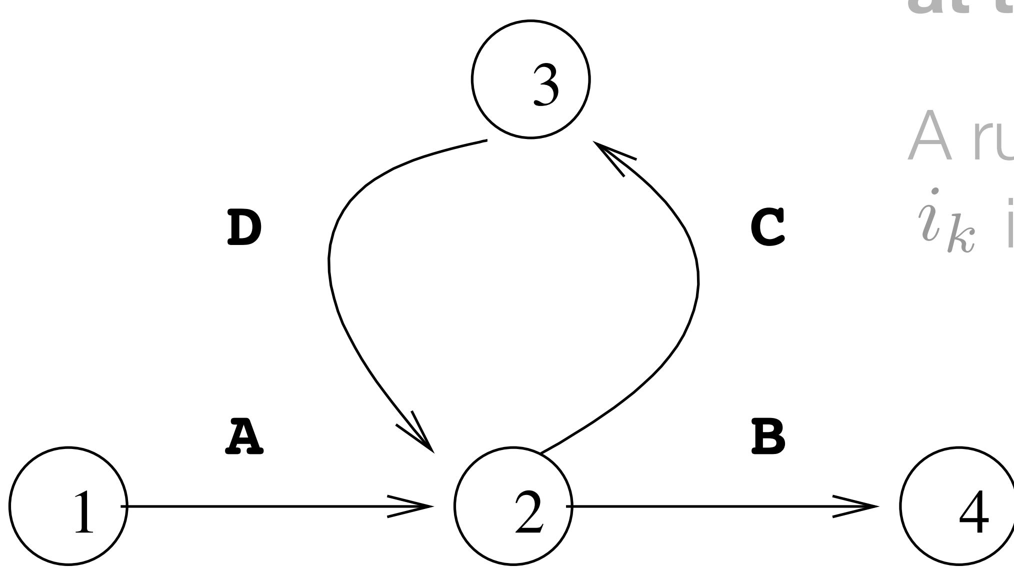
Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events
- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  This one, stops at i_k minimal (=8):

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑			↑	↑			↑							

We would like to be warned **at the earliest** possible time

1 4 5 8



A run is **minimal** iff i_k is minimal (w. i_1 fixed) and ...

Semantics, 2: «shortest runs»

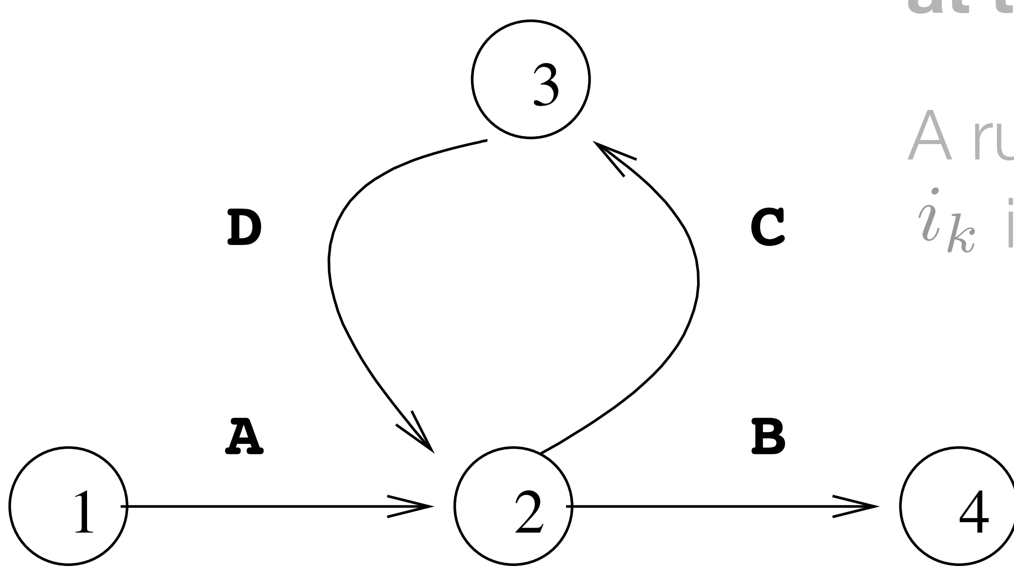
- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  And this one too:

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑			↑			↑	↑							

We would like to be warned **at the earliest** possible time

1 4 7 8



A run is **minimal** iff i_k is minimal (w. i_1 fixed) and ...

Semantics, 2: «shortest runs»

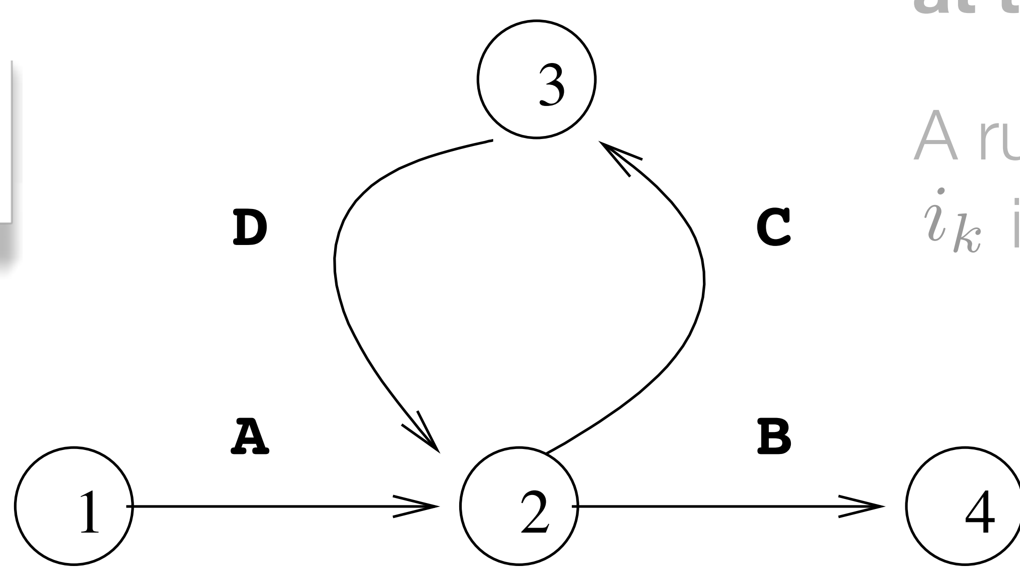
- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  And again this one!

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑	↑	↑	↑	↑	↑	↑	↑							

We would like to be warned **at the earliest** possible time

1 2 3 4 5 6 7 8



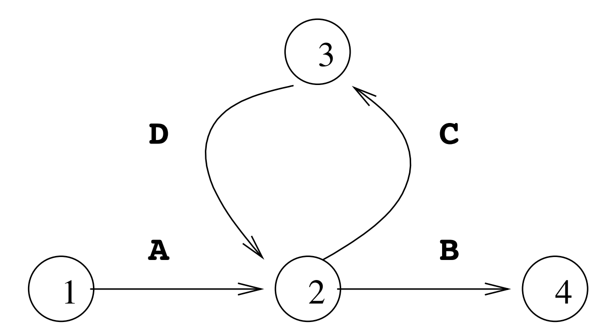
A run is **minimal** iff i_k is minimal (w. i_1 fixed) and ...

The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 8
1 2 3 8
1 2 5 8
1 2 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 4 5 6 7 8
1 2 3 4 5 6 7 8



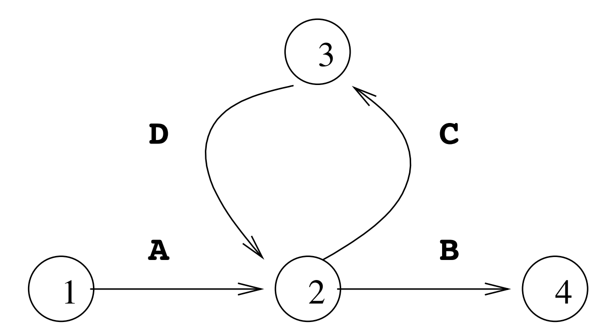
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 8
1 2 3 8
1 2 5 8
1 2 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 4 5 6 7 8
1 2 3 4 5 6 7 8



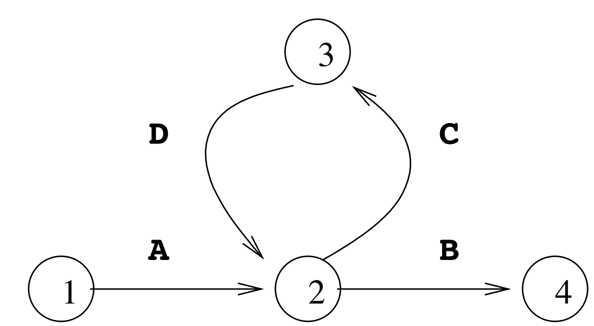
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 8
1 2 3 8
1 2 5 8
1 2 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 4 5 6 7 8
1 2 3 4 5 6 7 8

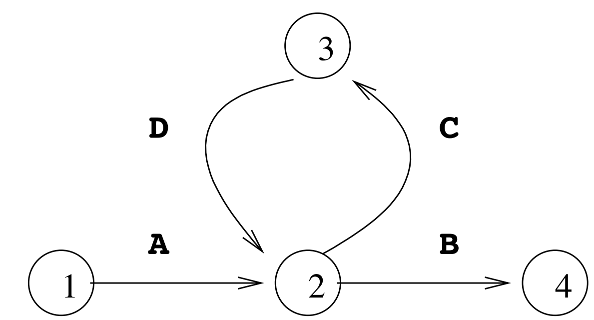


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 2 3 8
1 2 5 8
1 2 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 4 5 6 7 8
1 2 3 4 5 6 7 8
1 8

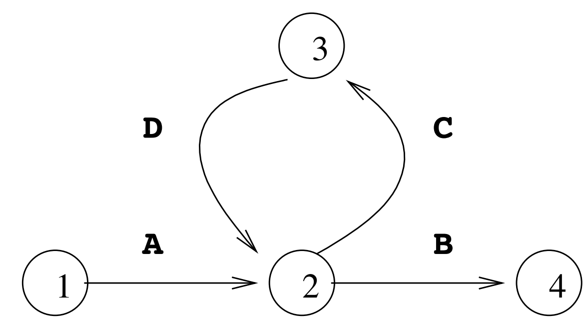


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let’s **sort** in increasing order

1 2 3 8
1 2 5 8
1 2 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 4 5 6 7 8
1 2 3 4 5 6 7 8
1 8



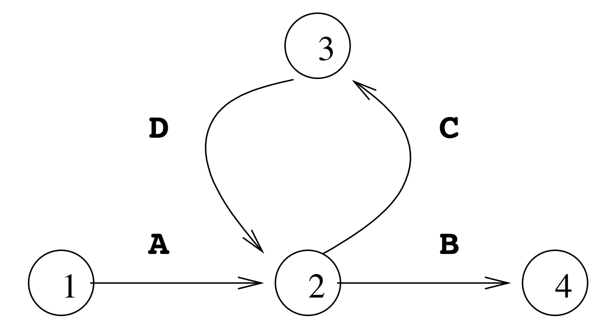
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 8
1 2 5 8
1 2 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 2 3 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 4 5 6 7 8
1 8

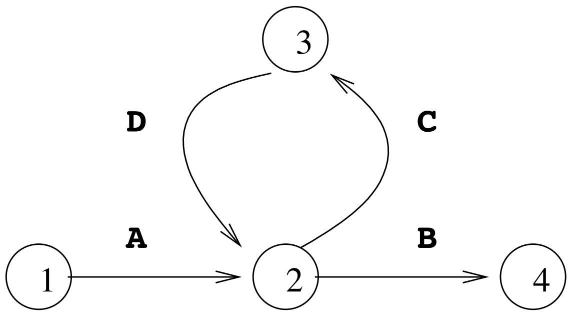


The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 8
1 2 5 8
1 2 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 2 3 4 5 6 7 8
1 4 5 8
1 4 7 8
1 4 5 6 7 8
1 6 7 8
1 8



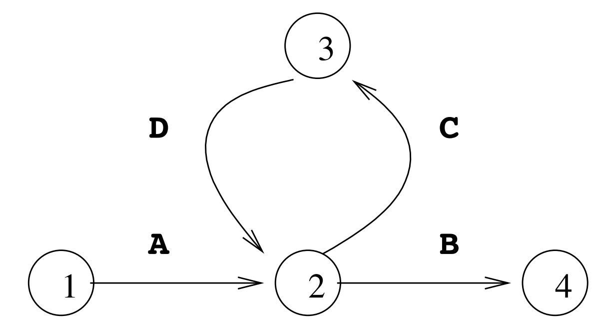
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 8
1 2 5 8
1 2 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 6 7 8
1 2 3 4 5 6 7 8
1 4 5 8
1 4 7 8
1 4 5 6 7 8
1 6 7 8
1 8



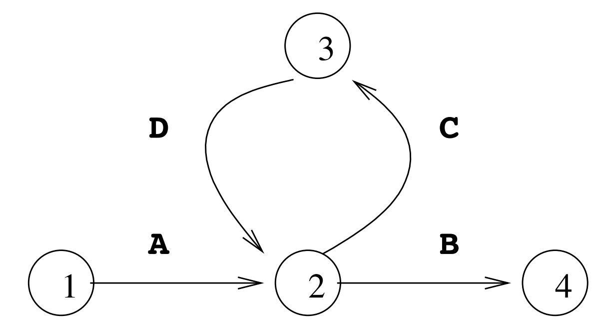
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 5 8
1 2 7 8
1 2 5 6 7 8
1 2 3 4 5 6 7 8
1 4 5 8
1 4 7 8
1 4 5 6 7 8
1 6 7 8
1 8



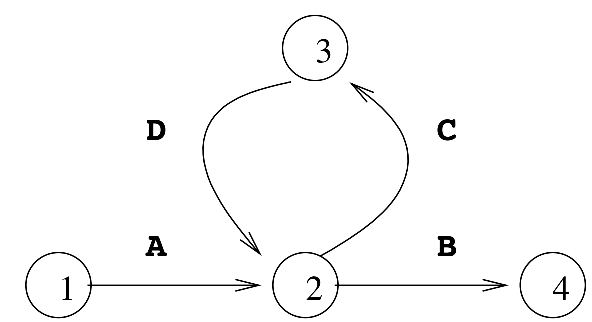
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 4 5 6 7 8
1 2 5 8
1 2 7 8
1 2 5 6 7 8
1 4 5 8
1 4 7 8
1 4 5 6 7 8
1 6 7 8
1 8



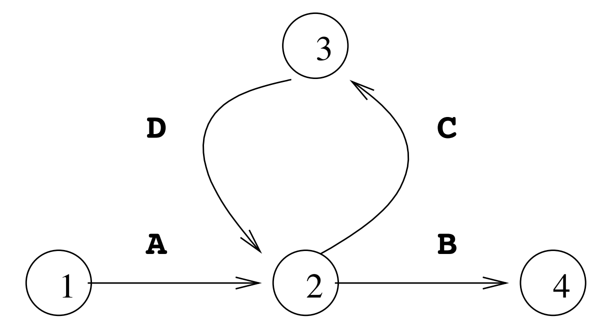
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 4 5 6 7 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 7 8
1 4 5 6 7 8
1 6 7 8
1 8



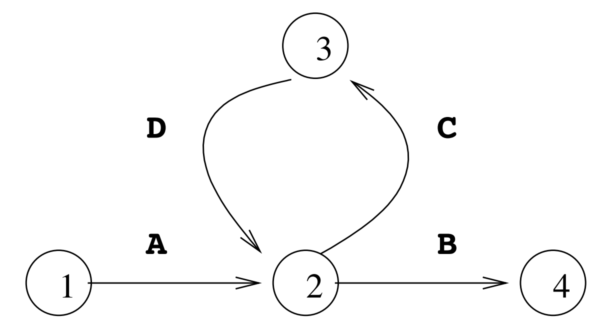
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 4 5 6 7 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8



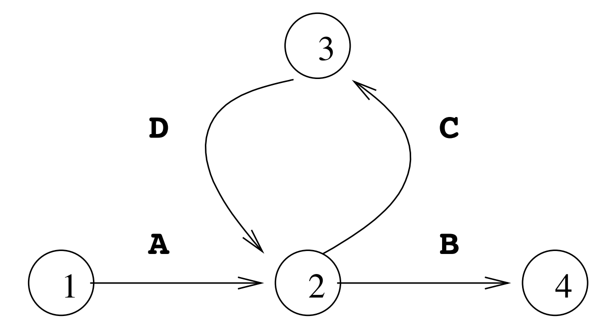
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let’s **sort** in increasing order

1 2 3 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 4 5 6 7 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8

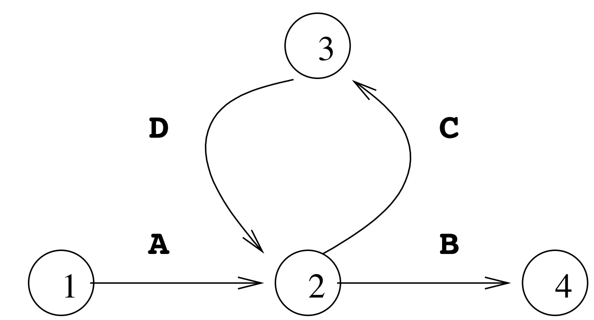


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 8
1 2 3 6 7 8
1 2 3 4 5 6 7 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8

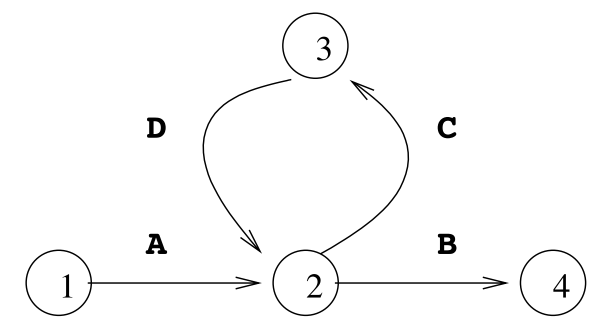


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 4 5 6 7 8
1 2 3 8
1 2 3 6 7 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8



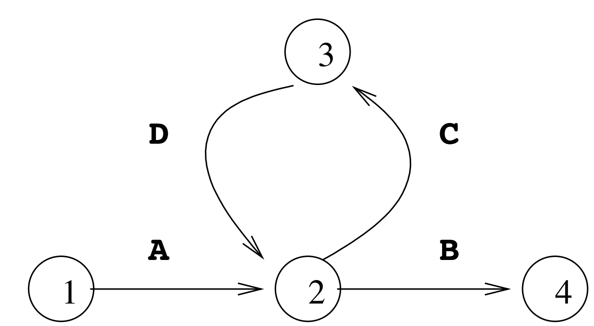
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 4 5 6 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 8
1 2 5 6 7 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8



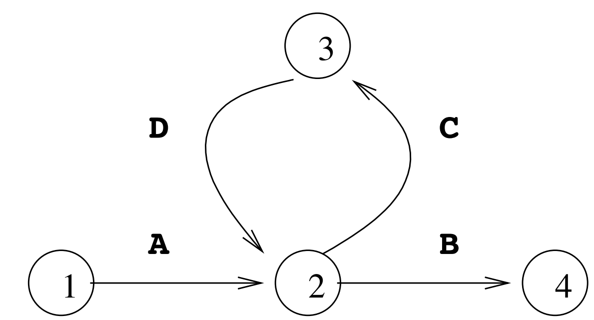
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 4 5 6 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 8
1 4 5 6 7 8
1 4 7 8
1 6 7 8
1 8



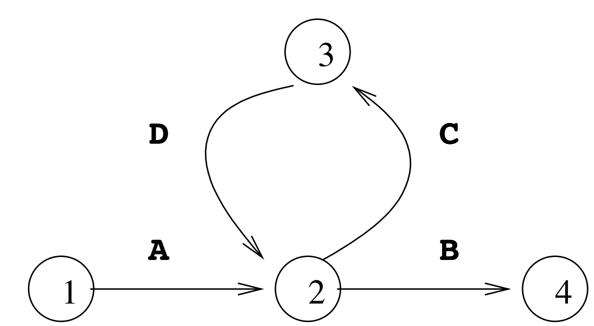
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 4 5 6 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



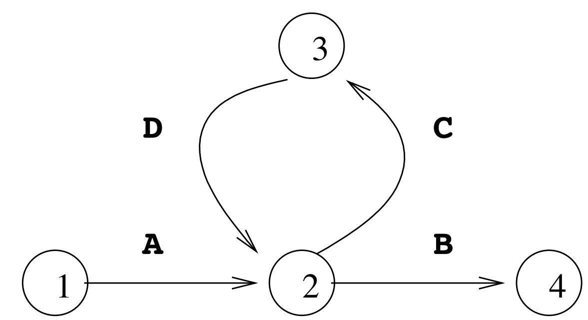
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 4 5 6 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



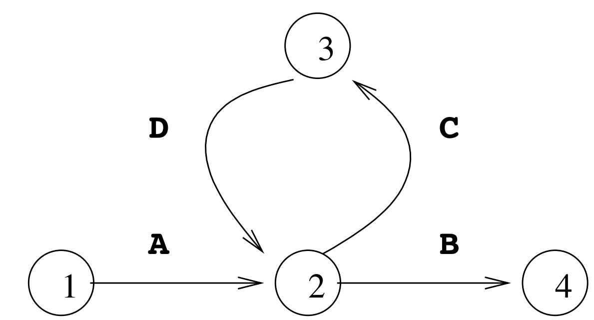
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 4 5 8
1 2 3 4 5 6 7 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



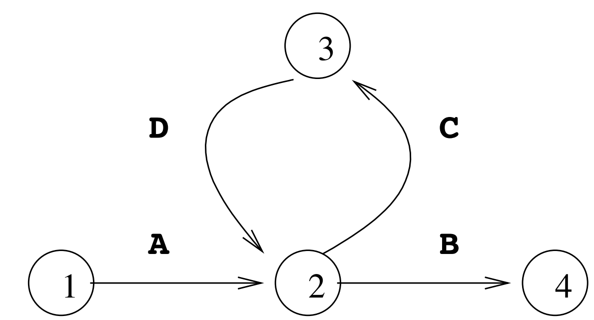
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 4 5 8
1 2 3 4 5 6 7 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8

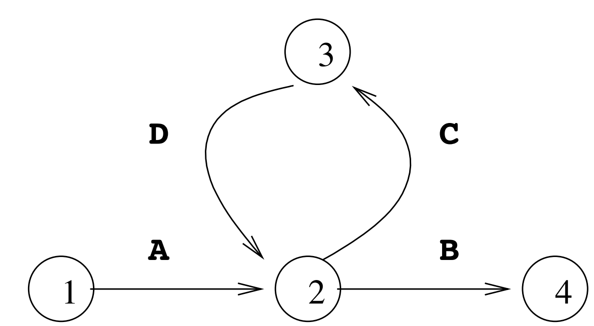


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



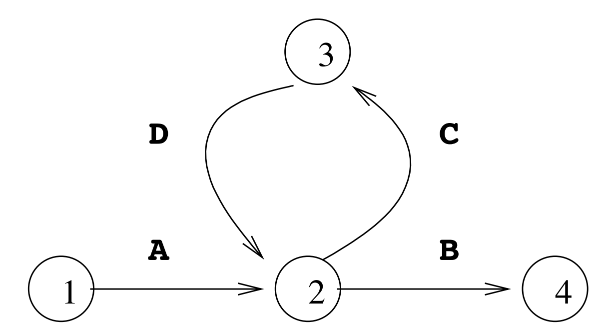
The lexicographic ordering

- ... or dictionary order
  but take indices instead of letters...

- and let’s **sort** in increasing order

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8

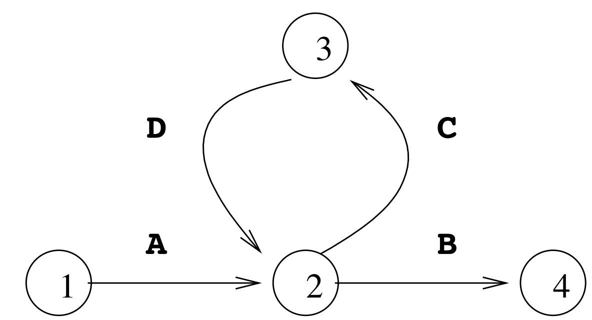


The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let's **sort** in increasing order

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



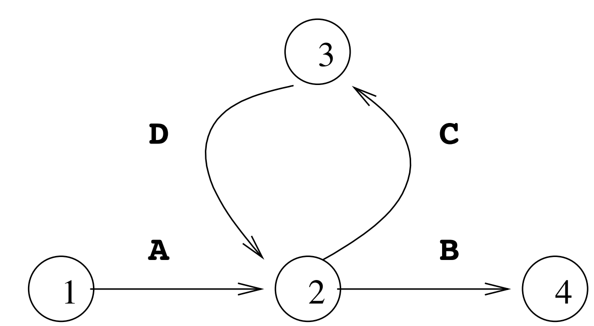
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



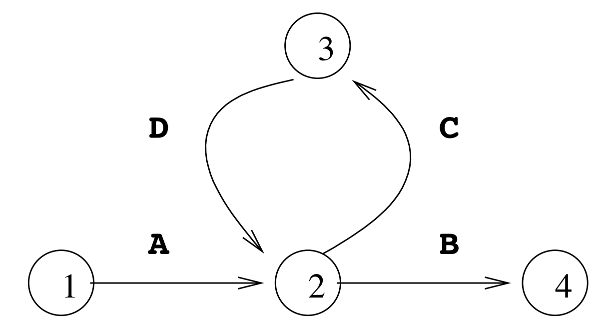
The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...

- and let's **sort** in increasing order

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



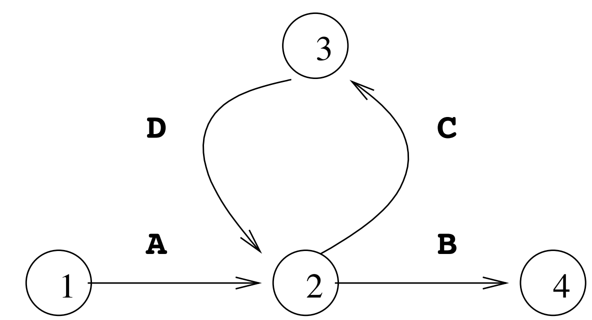
The **largest**

The lexicographic ordering

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- ... or dictionary order
  but take indices instead of letters...
- and let’s **sort** in increasing order

1 2 3 4 5 6 7 8
1 2 3 4 5 8
1 2 3 4 7 8
1 2 3 6 7 8
1 2 3 8
1 2 5 6 7 8
1 2 5 8
1 2 7 8
1 4 5 6 7 8
1 4 5 8
1 4 7 8
1 6 7 8
1 8



The **smallest**

... and most **informative**

The **largest**

Semantics, 2: «shortest runs»

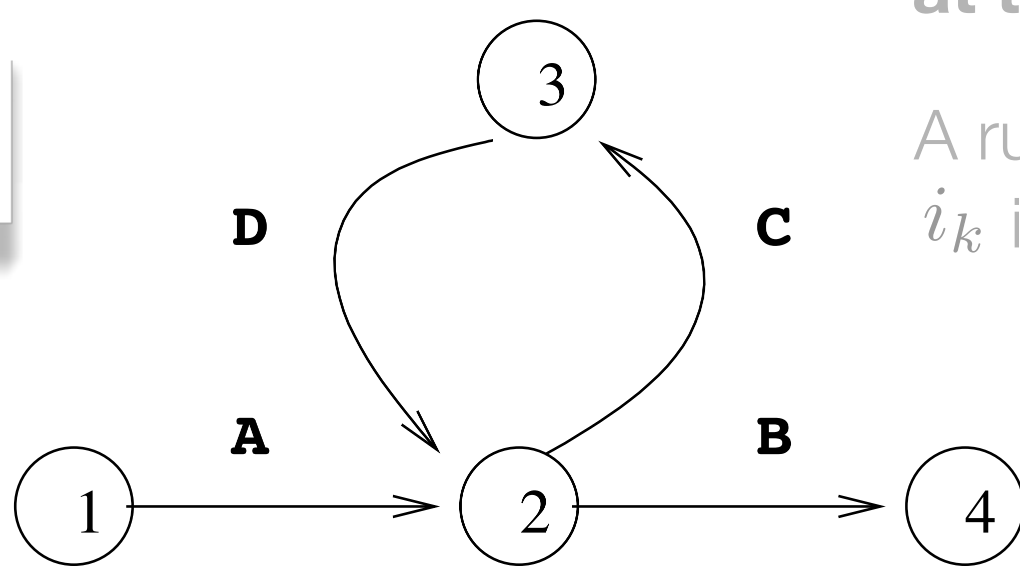
- ORCHIDS looks for subsequences of events
- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑	↑	↑	↑	↑	↑	↑	↑							

We would like to be warned **at the earliest** possible time

The minimal run:

1 2 3 4 5 6 7 8



A run is **minimal** iff i_k is minimal (w. i_1 fixed) and ...

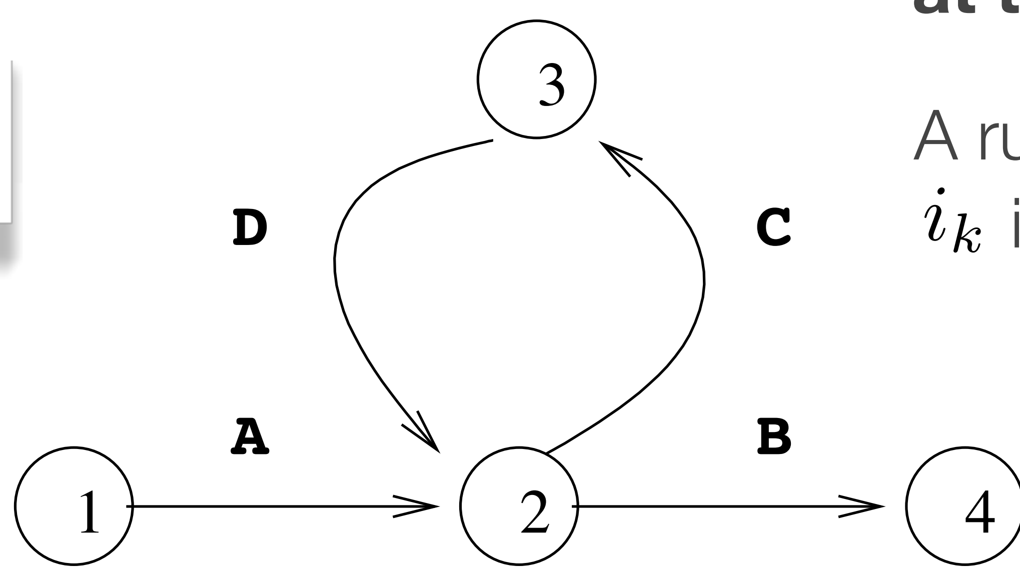
Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events
- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑	↑	↑	↑	↑	↑	↑	↑							

The minimal run:

1 2 3 4 5 6 7 8



We would like to be warned **at the earliest** possible time

A run is **minimal** iff i_k is minimal (w. i_1 fixed) and ...

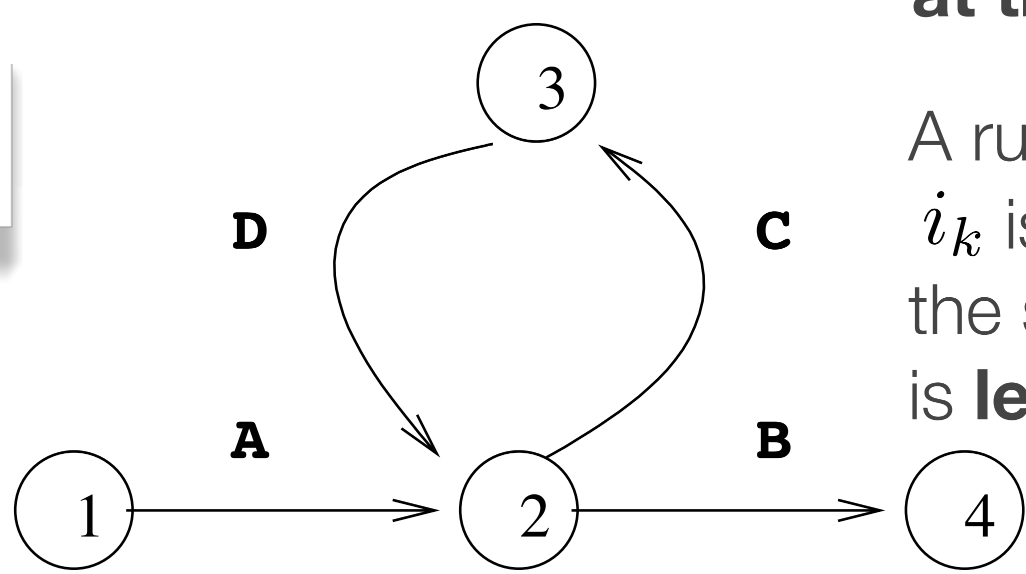
Semantics, 2: «shortest runs»

- ORCHIDS looks for subsequences of events
- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$

A	C	D	C	D	C	D	B	A	A	C	B	D	C	A
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↑	↑	↑	↑	↑	↑	↑	↑							

The minimal run:

1 2 3 4 5 6 7 8



We would like to be warned **at the earliest** possible time

A run is **minimal** iff i_k is minimal (w. i_1 fixed) and the sequence $i_1 < i_2 < \ldots < i_k$ is **lexicographically minimal**

Semantics => **Theorems**

- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  It is **minimal** iff i_k is minimal (w. i_1 fixed) and
  $i_1 < i_2 < \ldots < i_k$ is **lexicographically** smallest.

Proposition (optimality):
If there is a run starting at i_1 ,
then there is a **unique** one that is **minimal**.

Proof: the associated ordering on runs is
- — well-founded (whence existence)
- — total (whence uniqueness)

Semantics => **Theorems**

- ORCHIDS looks for subsequences of events

- A **run** is an increasing sequence of indices $i_1 < i_2 < \ldots < i_k$
  It is **minimal** iff i_k is minimal (w. i_1 fixed) and
  $i_1 < i_2 < \ldots < i_k$ is **lexicographically** smallest.

Proposition (optimality):
If there is a run starting at i_1 ,
then there is a **unique** one that is **minimal**.

Proof: the associated ordering on runs is
— well-founded (whence existence)
— total (whence uniqueness) □

Algorithms

- The ORCHIDS algorithm never sorts anything
- Instead, it **keeps** the thread queue **sorted** at all times
- ... for a subtle ordering: at event #n,

$$[i_1, i_2, \cdots, i_k] \leq_n [j_1, j_2, \cdots, j_\ell]$$

if and only if

$$i_1 = j_1 \text{ and}$$

$$[i_1, i_2, \cdots, i_k, n] \text{ lexicographically smaller than } [j_1, j_2, \cdots, j_\ell, n]$$

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Motto: keep queues sorted

old_queue | 1 2 3 | 1 2 – | 1 – 3 | thread

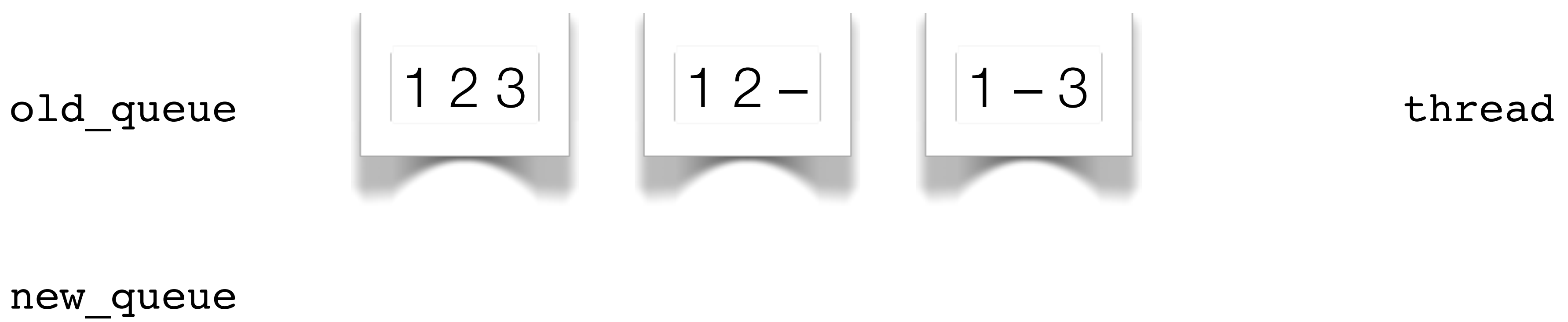
new_queue

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted



Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted

`old_queue` 1 2 – | 1 – 3

`thread` 1 2 3

`new_queue`

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted

old_queue: 1 2 – | 1 – 3

thread: 1 2 3

new_queue: 1 2 3 4

Algorithms

Read event #4

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Motto: keep queues sorted

`old_queue` 1 2 – 1 – 3 `thread`

`new_queue` 1 2 3 4 1 2 3 –

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted

`old_queue` 1 – 3

`thread` 1 2 –

`new_queue` 1 2 3 4 | 1 2 3 –

Algorithms

Read event #4

Motto: keep queues sorted

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

`old_queue` 1 – 3

`thread` 1 2 –

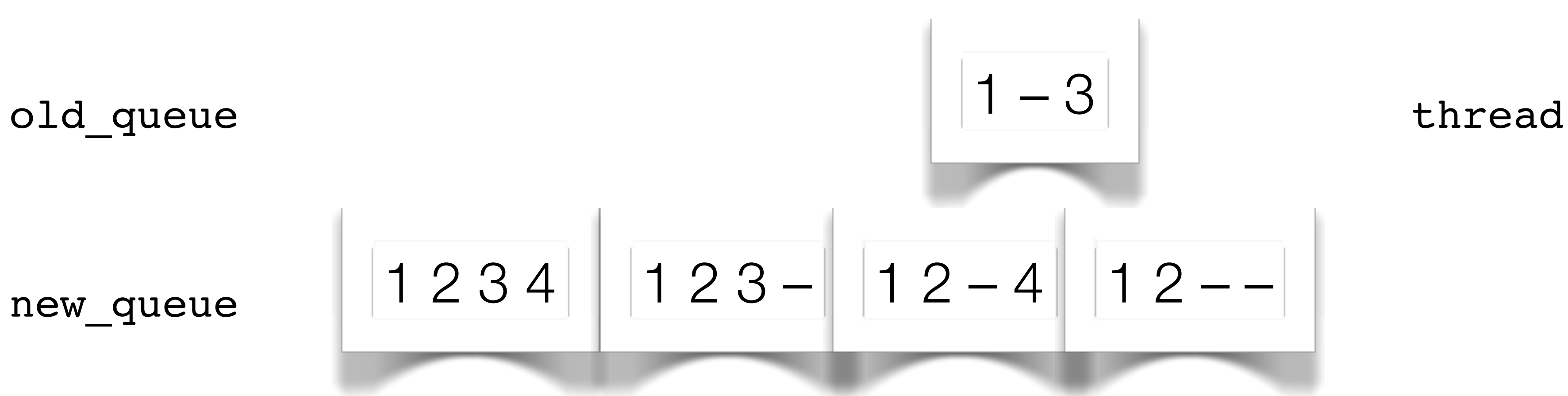
`new_queue` 1 2 3 4 | 1 2 3 – | 1 2 – 4

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted



Algorithms

Read event #4

Motto: keep queues sorted

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

`old_queue`

`thread` 1 – 3

`new_queue` 1 2 3 4 | 1 2 3 – | 1 2 – 4 | 1 2 – –

Algorithms

Read event #4

Motto: keep queues sorted

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

old_queue

thread: 1 – 3

new_queue: 1 2 3 4 | 1 2 3 – | 1 2 – 4 | 1 2 – – | 1 – 3 4

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

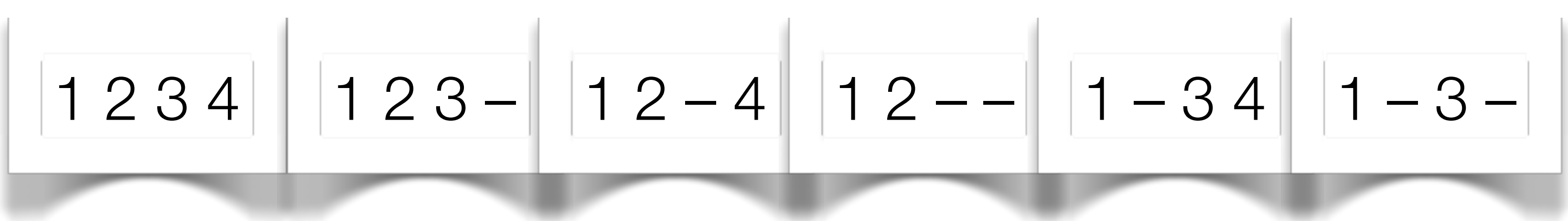
Read event #4

Motto: keep queues sorted

old_queue

thread

new_queue



Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted

old_queue

thread

new_queue

1 2 3 4	1 2 3 –	1 2 – 4	1 2 – –	1 – 3 4	1 – 3 –	4

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #4

Motto: keep queues sorted

old_queue: 1 2 3 4 | 1 2 3 – | 1 2 – 4 | 1 2 – – | 1 – 3 4 | 1 – 3 – | 4

new_queue

Algorithms

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Read event #5

Motto: keep queues sorted

old_queue | 1 2 3 4 | 1 2 3 – | 1 2 – 4 | 1 2 – – | 1 – 3 4 | 1 – 3 – | 4

new_queue

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**.

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4

```
e = next_event();
new_queue = empty();
while (thread = dequeue (old_queue)) {
    if (eval_guard (g, e)) {
      execute_action (a);
  enqueue (new_queue, thread);
}
for each rule r do enqueue (new_queue, r->init);
old_queue = new_queue;
```

`old_queue` 1 – 3 `thread`

`new_queue`

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g/a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

`old_queue` `thread` 1 – 3

`new_queue`

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4

```
orchids main loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

`old_queue`

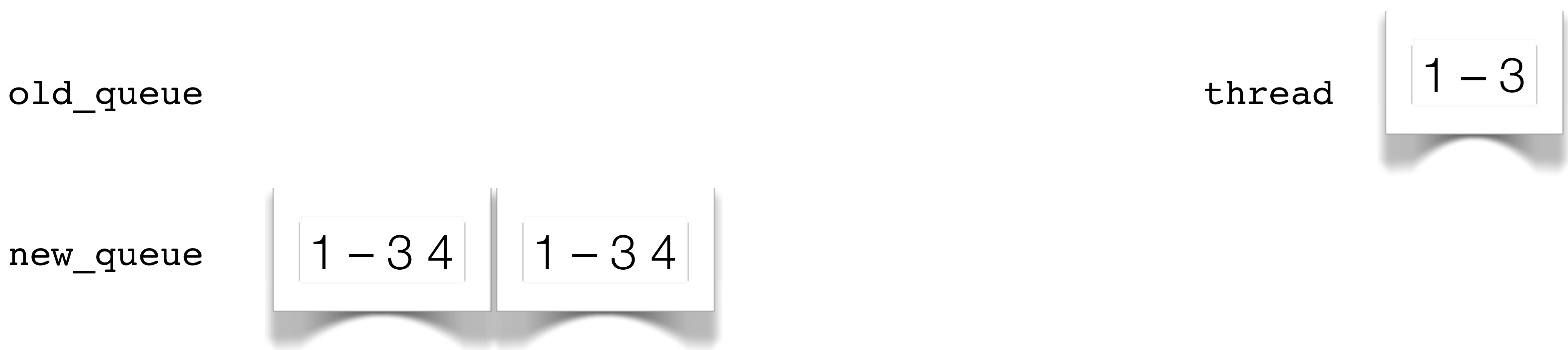
`thread` 1 – 3

`new_queue` 1 – 3 4

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4

```
orchids main loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread -g/a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```



Algorithms (?)

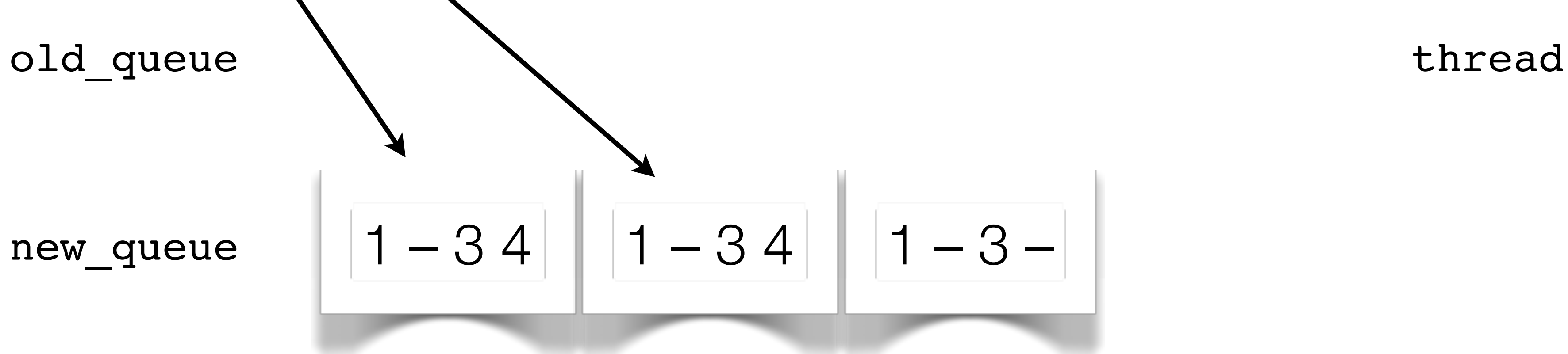
- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  while (thread = dequeue (old_queue)) {
    for each outgoing transition [thread –g/a→ t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (new_queue, t);
      }
    enqueue (new_queue, thread);
  }
  for each rule r do enqueue (new_queue, r->init);
  old_queue = new_queue;
```

`old_queue` `thread`

Algorithms (?)

- Several optimizations, avoiding exponential blow-up in most cases
- Main problem: the latter algorithm is **wrong**:
- Imagine we now have **two** outgoing transitions at event 4
  the first one will raise an alert at 1 – 3 4 – 6
  the second one will raise an alert at 1 – 3 4 5 6

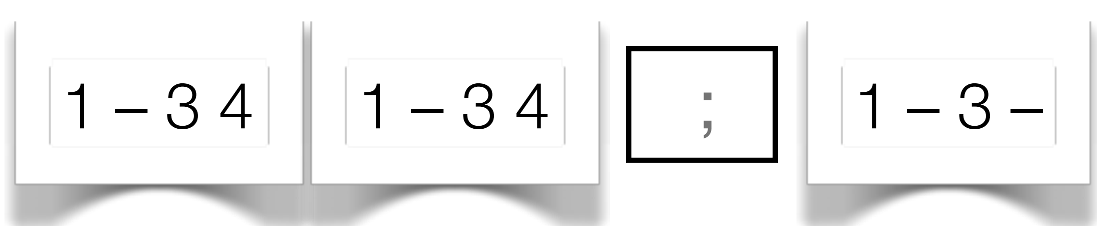


Fixing the bug

- Instead of lists of threads, encode queues as lists of **blobs**, where a blob is an unsorted list of threads with the same sequence of events

unsorted

- Practical implementation: use fake thread «;»



unsorted

Algorithms: the right one

```
orchids_main_loop:
  e = next_event();
  new_queue = empty();
  unsorted = empty(); next = empty();
  while (thread = dequeue (old_queue)) {
    if (thread == «;») bump() else
    for each outgoing transition [thread -g,a-> t] do
      if (eval_guard (g, e)) {
         execute_action (a);
         enqueue (unsorted, t);
      }
    enqueue (next, thread);
  }
  bump();
  for each rule r do
    enqueue (new_queue, r->init);
  bump();
  old_queue = new_queue;
```

```
bump:
  enqueue_all (new_queue, unsorted);
  unsorted = empty();
  enqueue (new_queue, «;»);
  enqueue_all (new_queue, next);
  next = empty();
  enqueue (new_queue, «;»);
```

```
/* Optimization: don't enqueue «;» if last element on queue is «;» already. */
```

Algorithms

- ORCHIDS looks for subsequences of events: **runs**
- Our algorithm finds these minimal runs by an **efficient** algorithm
  ... which, notably, never **sorts** anything

Theorem (soundness):
The ORCHIDS algorithm computes exactly the minimal runs.

Proof: slightly more complex (omitted).

[GO08] J. GOUBAULT-LARRECQ and J. OLIVAIN. A Smell of Orchids. *In* RV'08, LNCS 5289, pages 1-20. Springer, 2008. (PDF | BibTeX + Abstract)

Proof. Assume that $B'_0, B'_1, B'_2, \ldots, B'_{2m-1}, B'_{2m}$ is not $\leq_{i+1}$-sorted. Let D'_j be the subflow of B'_j, for all j, and D_j be the subflow of B_j. Then there are j', k' with $0 \leq k' < j' \leq 2m$ and $D'_{j'} \leq_{i+1} D'_{k'}$. Note that $k' \neq 0$, since the birthdate of any partial run in B'_0 is $i+1$, which is different from all other birthdates. Write $k' = 2k - \delta_k$ and $j' = 2j - \delta_j$, where δ_k, δ_j are 0 or 1, and $k \leq j$. If $k = j$, then $k' < j'$ implies $\delta_k = 1, \delta_j = 0$, so that $D'_{k'} = D_k \cup \{i+1\}$ (the partial runs of $B'_{k'} = B'_{2k-1}$ are non-trivial extensions of those of B_k), and $D'_{j'} = D_k$ (those of $B'_{j'} = B'_{2j} = B'_{2k}$ are trivial extensions). But $D_k \cup \{i+1\} <_{i+1} D_k$, so $D'_{k'} <_{i+1} D'_{j'}$, contradiction.

So $k < j$. Then $D_{k'}$ equals D_k, possibly with $i+1$ added, and $D_{j'}$ equals D_j, possibly with $i+1$ added. Since $B_1, B_2, \ldots, B_m$ is $\leq_i$-sorted, it is impossible that $D_j \leq_i D_k$, i.e., that $D_j \cup \{i+1\} \leq_{lex} D_k \cup \{i+1\}$. Since $\leq_{lex}$ is a total ordering, we must have $D_k \cup \{i+1\} <_{lex} D_j \cup \{i+1\}$. Write the elements of D_k as $i_1 < i_2 < \ldots < i_p$ (with $i_p < i+1$), those of D_j as $j_1 < j_2 < \ldots < j_q$ (with $j_q < i+1$, and $j_1 = i_1$). Let $i_{p+1} = i+1$, $j_{q+1} = i+1$. Since $D_k \cup \{i+1\} <_{lex} D_j \cup \{i+1\}$, for some ℓ between 1 and $\min(p+1, q+1)$, $i_1 = j_1, i_2 = j_2, \ldots, i_{\ell-1} = j_{\ell-1}$, and $i_\ell < j_\ell$. Now $\ell \neq p+1$, else $i+1 = i_\ell < j_\ell \leq j_{q+1} = i+1$. So $\ell \leq p$. But then $D_{k'} \cup \{i+2\}$, which is composed of $i_1, i_2, \ldots, i_p$ (optionally $i_{p+1} = i+1$) and $i+2$, is lexicographically smaller than $D_{j'} \cup \{i+2\}$, which is composed of $j_1, j_2, \ldots, j_q$ (optionally $j_{q+1} = i+1$) and $i+2$. That is, $D_{k'} <_{i+1} D_{j'}$, contradiction. □

Algorithms

- ORCHIDS looks for subsequences of events: **runs**
- Our algorithm finds these minimal runs by an **efficient** algorithm
  ... which, notably, never **sorts** anything

Theorem (soundness):
The ORCHIDS algorithm computes exactly the minimal runs.

Proof: slightly more complex (omitted).

☐

Proof. Assume that $B'_0, B'_1, B'_2, \ldots, B'_{2m-1}, B'_{2m}$ is not $\leq_{i+1}$-sorted. Let D'_j be the subflow of B'_j, for all j, and D_j be the subflow of B_j. Then there are j', k' with $0 \leq k' < j' \leq 2m$ and $D'_{j'} \leq_{i+1} D'_{k'}$. Note that $k' \neq 0$, since the birthdate of any partial run in B'_0 is $i+1$, which is different from all other birthdates. Write $k' = 2k - \delta_k$ and $j' = 2j - \delta_j$, where δ_k, δ_j are 0 or 1, and $k \leq j$. If $k = j$, then $k' < j'$ implies $\delta_k = 1, \delta_j = 0$, so that $D'_{k'} = D_k \cup \{i+1\}$ (the partial runs of $B'_{k'} = B'_{2k-1}$ are non-trivial extensions of those of B_k), and $D'_{j'} = D_k$ (those of $B'_{j'} = B'_{2j} = B'_{2k}$ are trivial extensions). But $D_k \cup \{i+1\} <_{i+1} D_k$, so $D'_{k'} <_{i+1} D'_{j'}$, contradiction.

So $k < j$. Then $D_{k'}$ equals D_k, possibly with $i+1$ added, and $D_{j'}$ equals D_j, possibly with $i+1$ added. Since $B_1, B_2, \ldots, B_m$ is $\leq_i$-sorted, it is impossible that $D_j \leq_i D_k$, i.e., that $D_j \cup \{i+1\} \leq_{lex} D_k \cup \{i+1\}$. Since $\leq_{lex}$ is a total ordering, we must have $D_k \cup \{i+1\} <_{lex} D_j \cup \{i+1\}$. Write the elements of D_k as $i_1 < i_2 < \ldots < i_p$ (with $i_p < i+1$), those of D_j as $j_1 < j_2 < \ldots < j_q$ (with $j_q < i+1$, and $j_1 = i_1$). Let $i_{p+1} = i+1$, $j_{q+1} = i+1$. Since $D_k \cup \{i+1\} <_{lex} D_j \cup \{i+1\}$, for some ℓ between 1 and $\min(p+1, q+1)$, $i_1 = j_1, i_2 = j_2, \ldots, i_{\ell-1} = j_{\ell-1}$, and $i_\ell < j_\ell$. Now $\ell \neq p+1$, else $i+1 = i_\ell < j_\ell \leq j_{q+1} = i+1$. So $\ell \leq p$. But then $D_{k'} \cup \{i+2\}$, which is composed of $i_1, i_2, \ldots, i_p$ (optionally $i_{p+1} = i+1$) and $i+2$, is lexicographically smaller than $D_{j'} \cup \{i+2\}$, which is composed of $j_1, j_2, \ldots, j_q$ (optionally $j_{q+1} = i+1$) and $i+2$. That is, $D_{k'} <_{i+1} D_{j'}$, contradiction. ☐

[GO08] J. GOUBAULT-LARRECQ and J. OLIVAIN. A Smell of Orchids. *In* RV'08, LNCS 5289, pages 1-20. Springer, 2008. (PDF | BibTeX + Abstract)

Algorithms

- ORCHIDS looks for subsequences of events: **runs**
- Our algorithm finds these minimal runs by an **efficient** algorithm
  ... which, notably, never **sorts** anything

Corollary (soundness and optimality):
1. ORCHIDS emits an alert at i_1 only if some run starts there
2. If there is a run starting at i_1 ,
   ORCHIDS emits only one alert, witnessing the minimal run.

Guarantees:
1. **no false positive**
2. absolute minimum «**information glut**» (at most 1 alert)
   and **no false negative** (at least 1 alert)

(in our model; the real world has its own perks, too)

Semantics, and optimizations

The «shortest runs» semantics also allows us to:

- **kill** threads which **provably** will **never find a run**
- **kill** threads which may ultimately find runs, which **provably cannot be minimal**
- ... by abstract interpretation techniques

[GO08] J. GOUBAULT-LARRECQ and J. OLIVAIN. A Smell of Orchids. *In* RV'08, LNCS 5289, pages 1-20. Springer, 2008. (PDF | BibTeX + Abstract)

- allowing for increased (time and space) **efficiency**

Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

Outline

1. A few **scary stories** about computer security
2. **ORCHIDS**: an intrusion prevention system
3. **Semantics** and algorithms
4. **NetEntropy**: detecting subverted cryptographic flows
5. Conclusion

The mod_ssl remote-to-local attack (McDonald 2003)

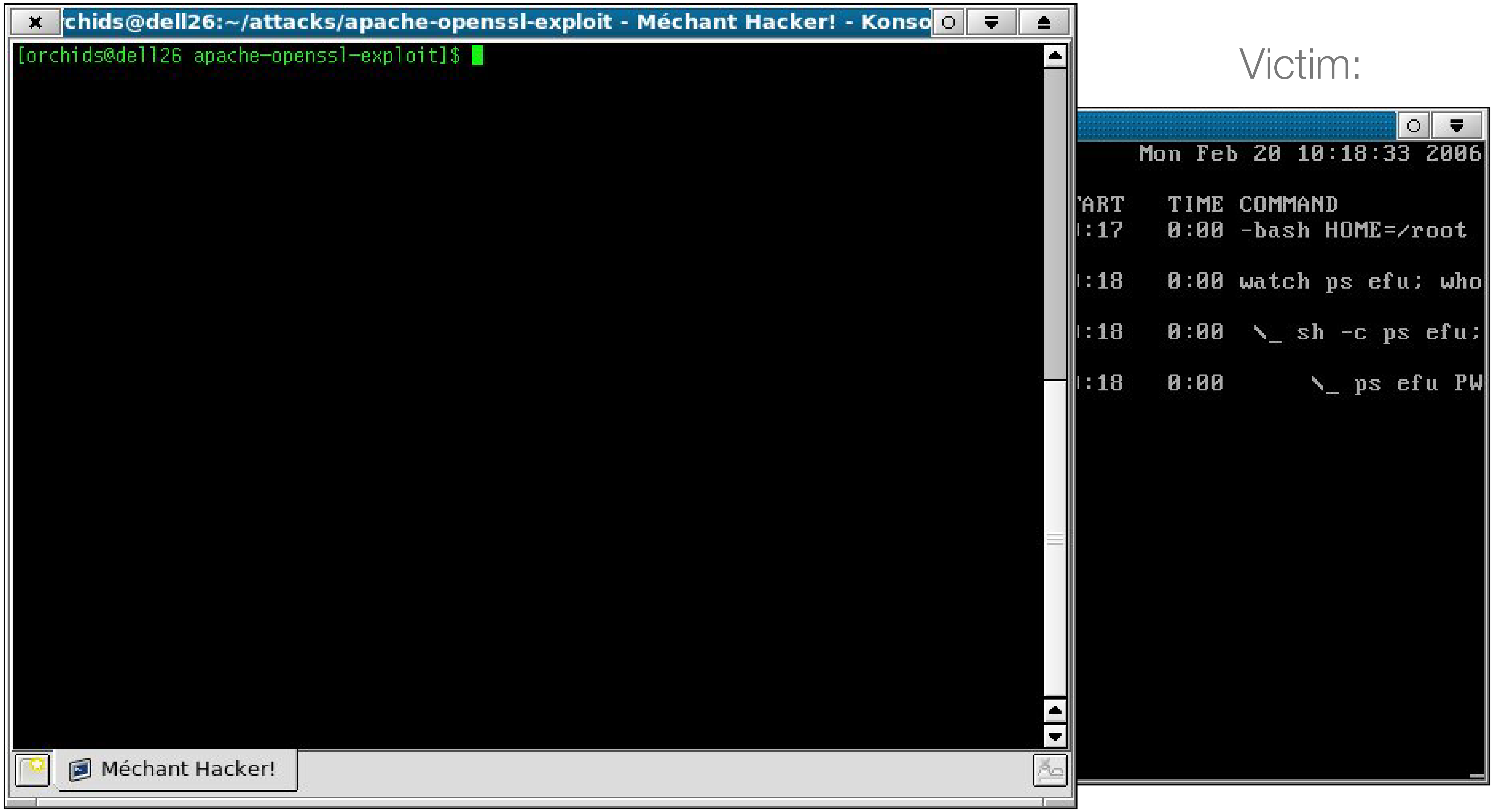
- ORCHIDS is not just a HIPS
- ORCHIDS does **anomaly**, too, not just **misuse** detection
- A **challenging** attack to detect:
  replaces **encrypted, random keys**
  by its own payload

  How do we detect illicit changes in **encrypted traffic**?

The mod_ssl remote-to-local attack (McDonald 2003)

Victim:

Remote attacker:

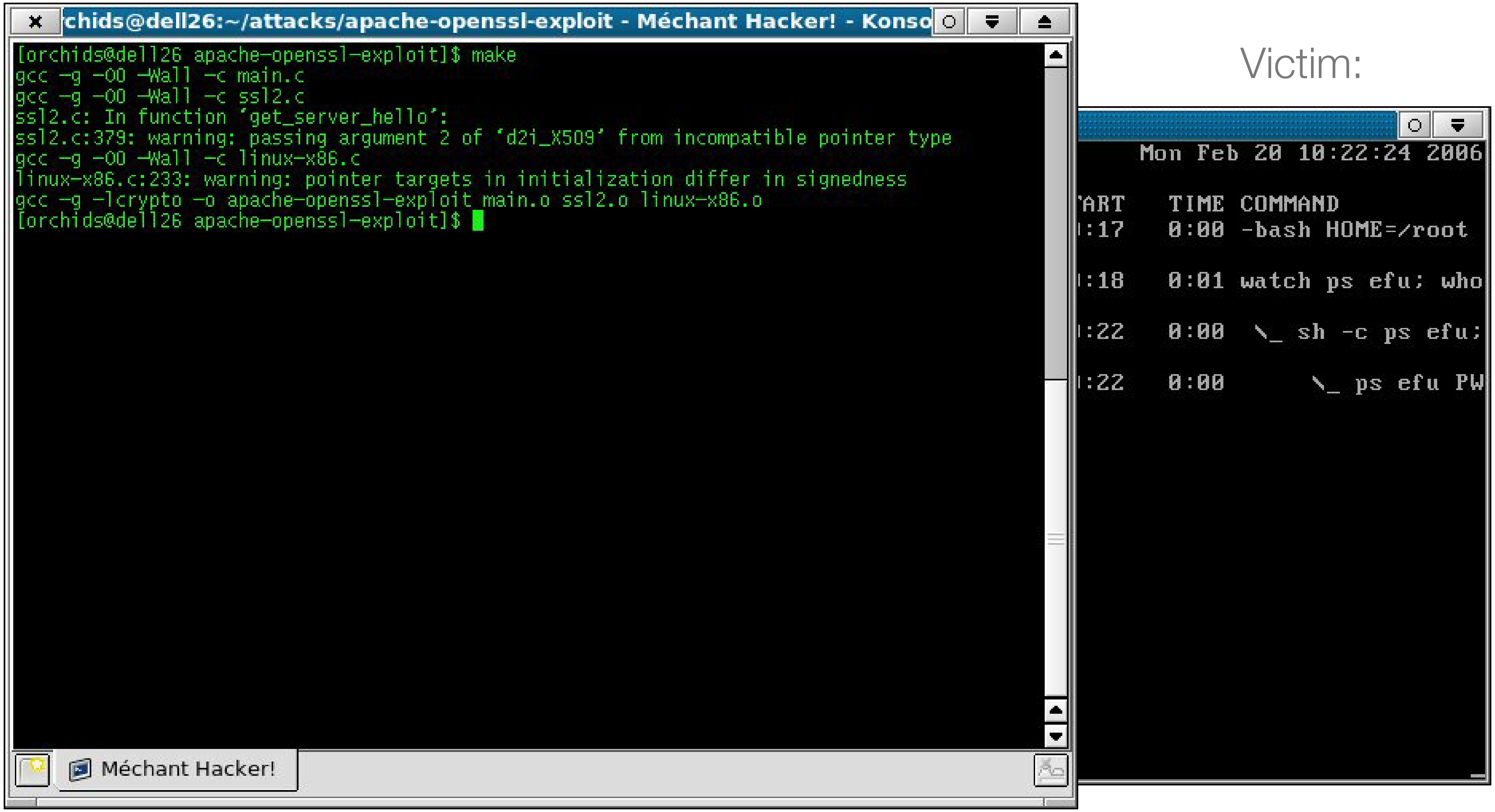


Compile attack: `apache-openssl-exploit`

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

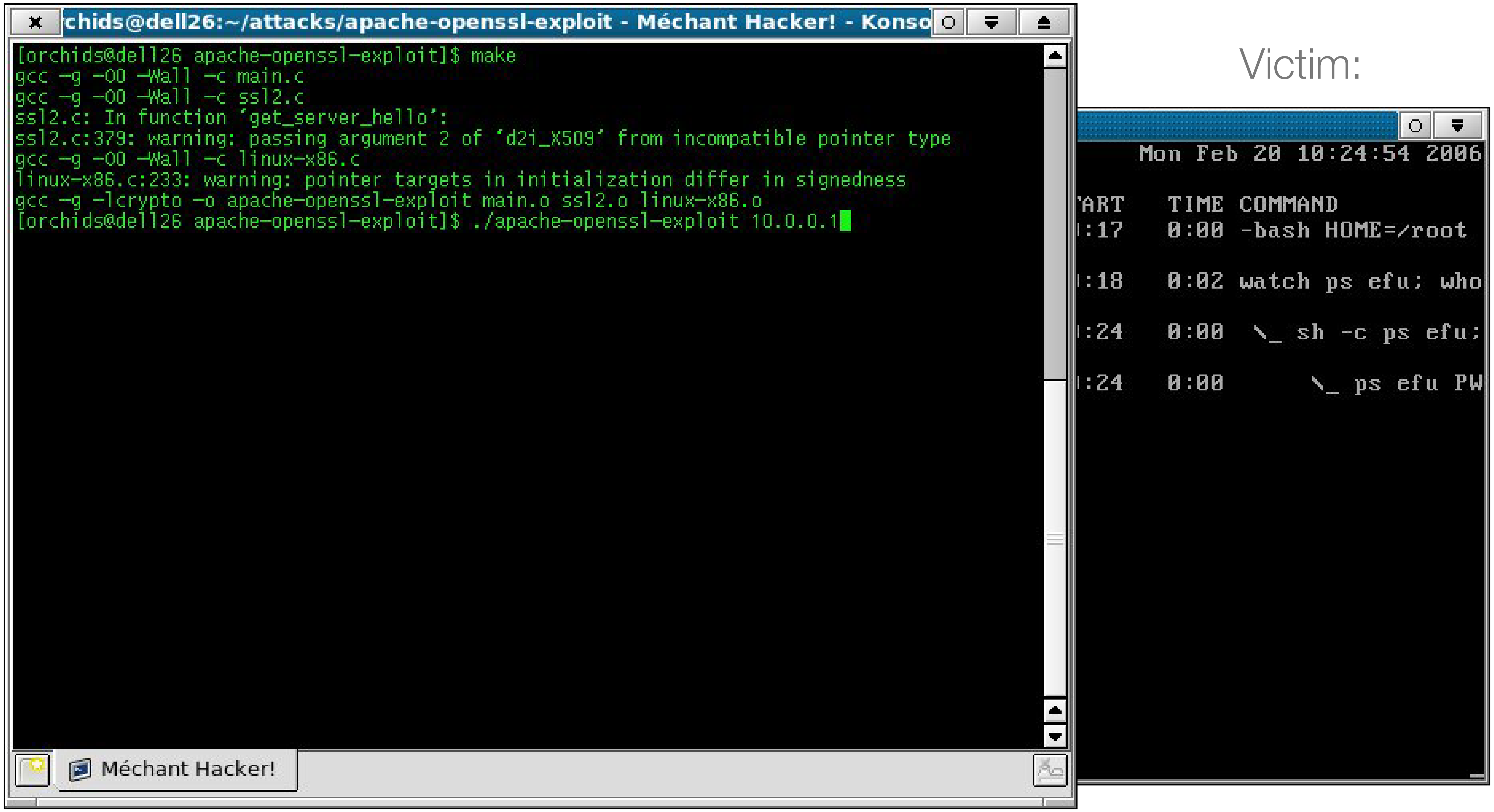
Victim:



The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

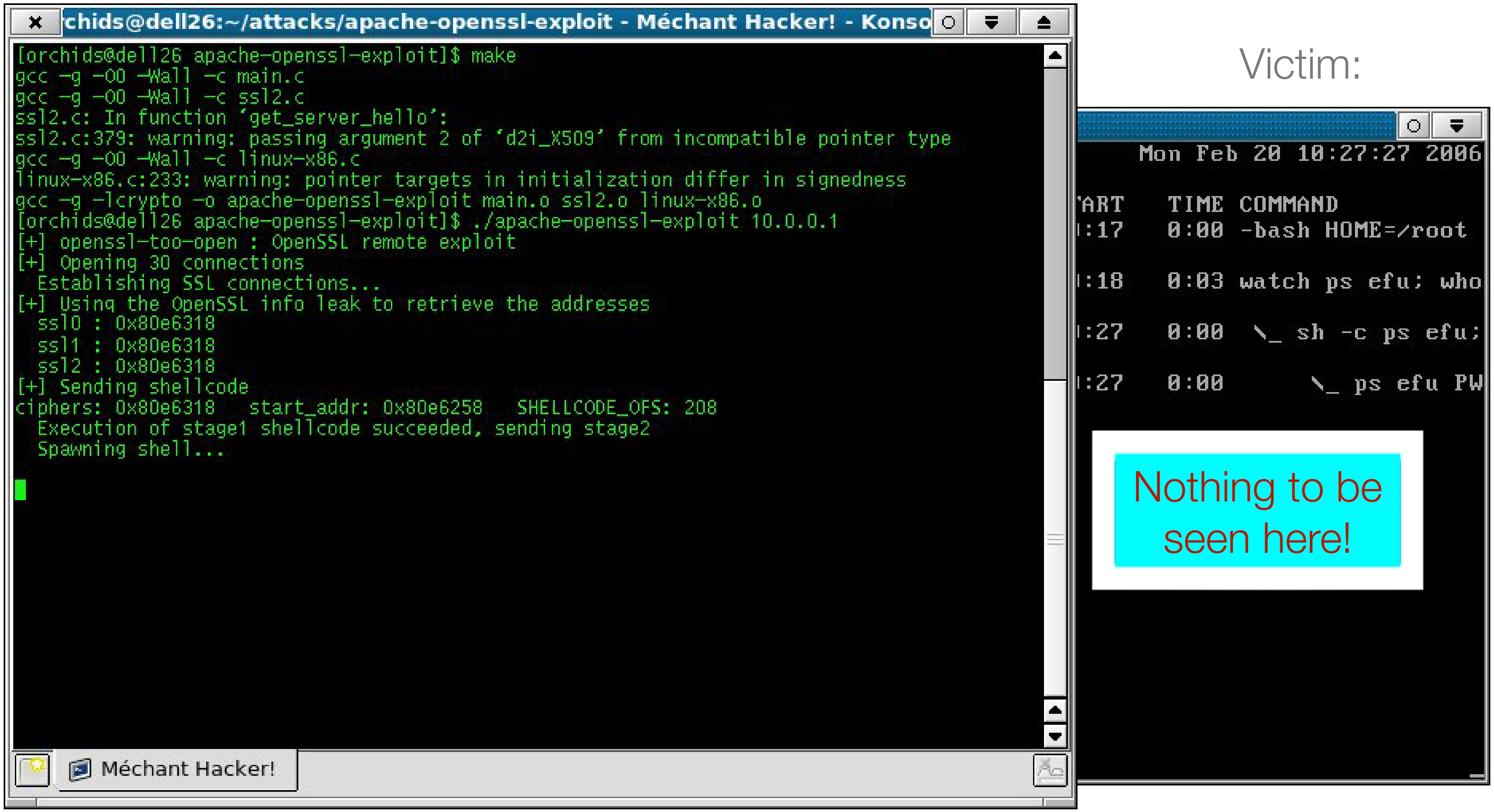


Launch attack: `apache-openssl-exploit`

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

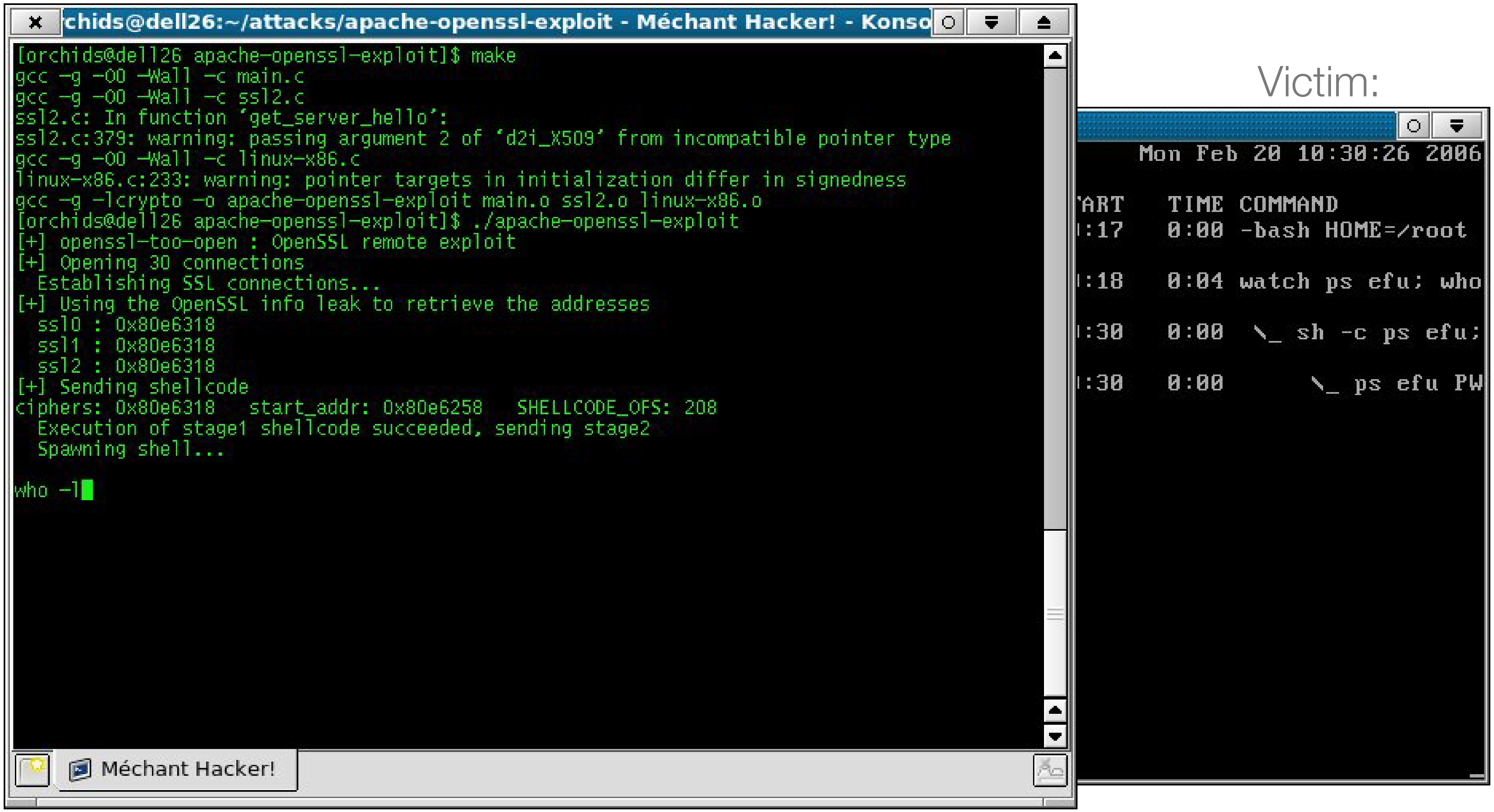


Success! The attacker connects to the victim machine.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

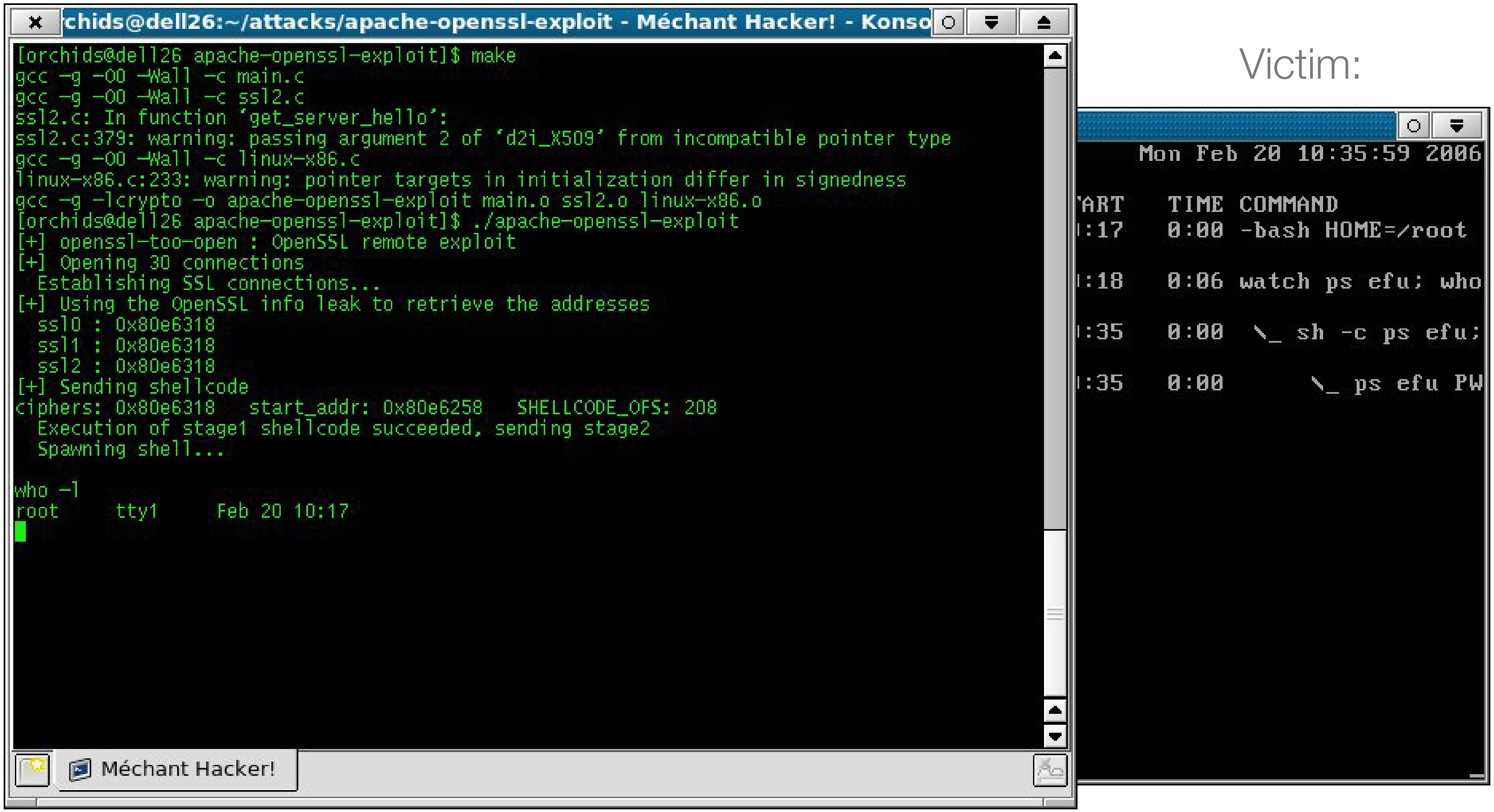


Check that it works...

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

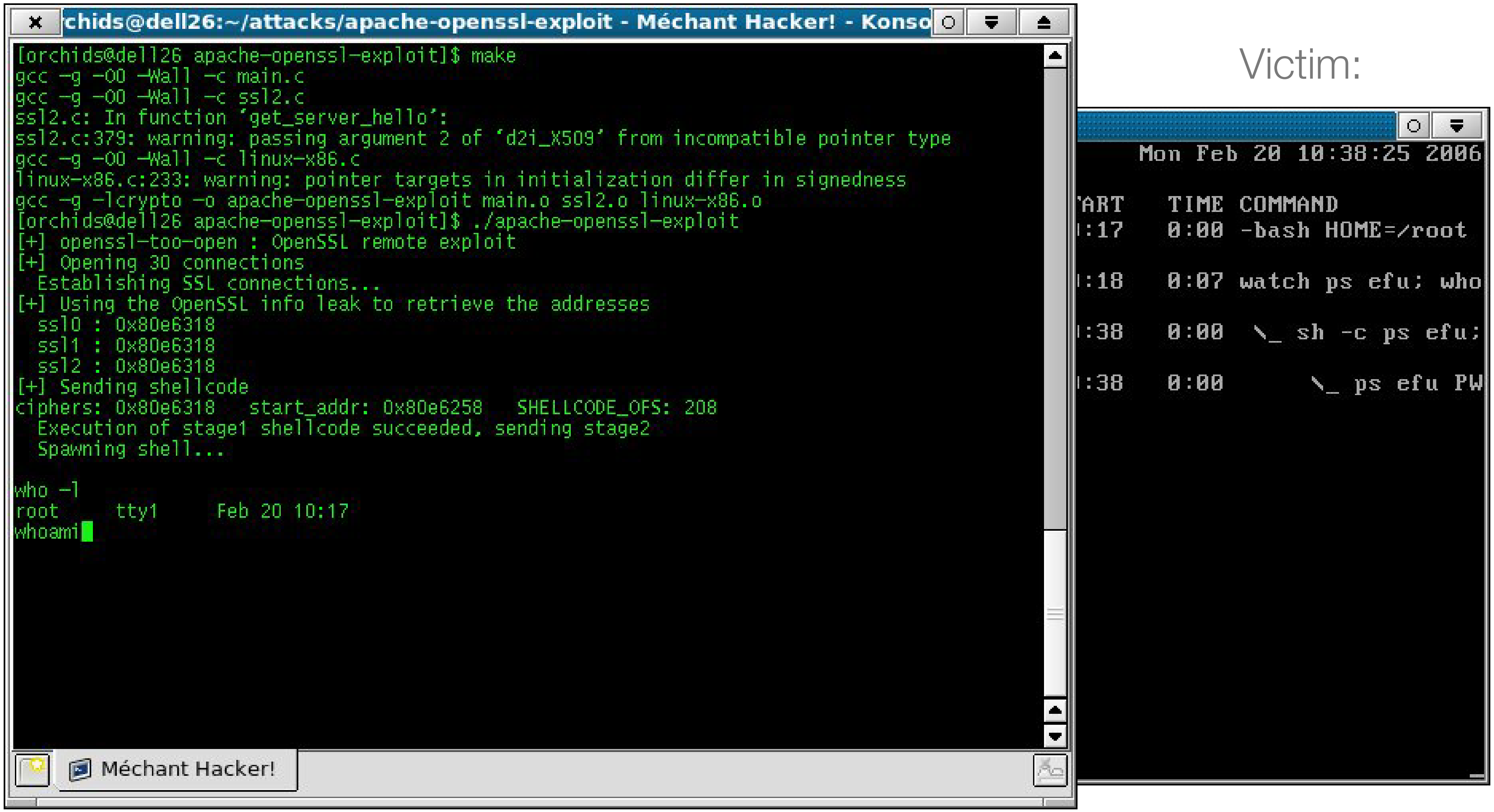


Works. Only `root` appears to be here (I am invisible...)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

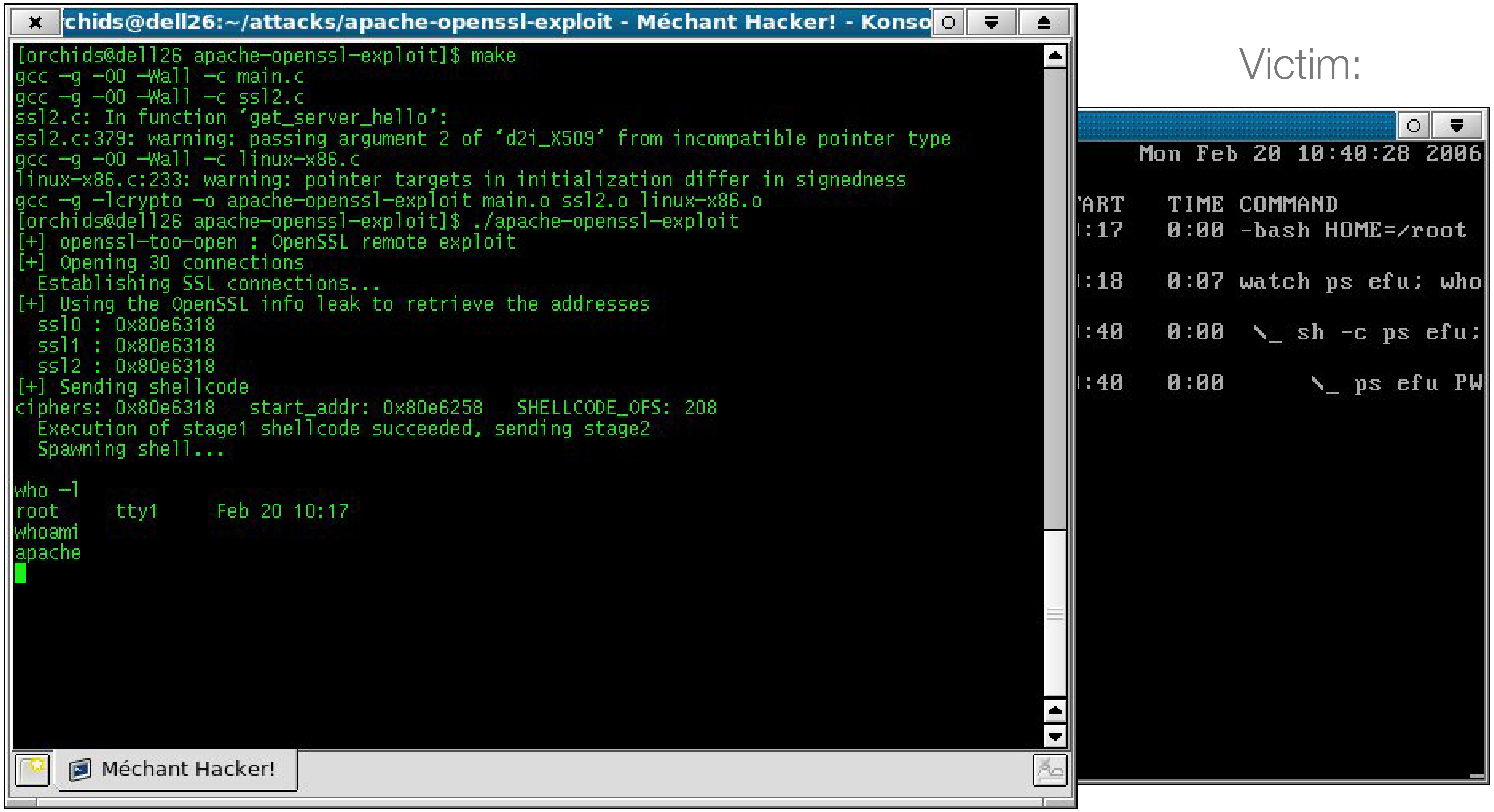


Works. Only `root` appears to be here (I am invisible...)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

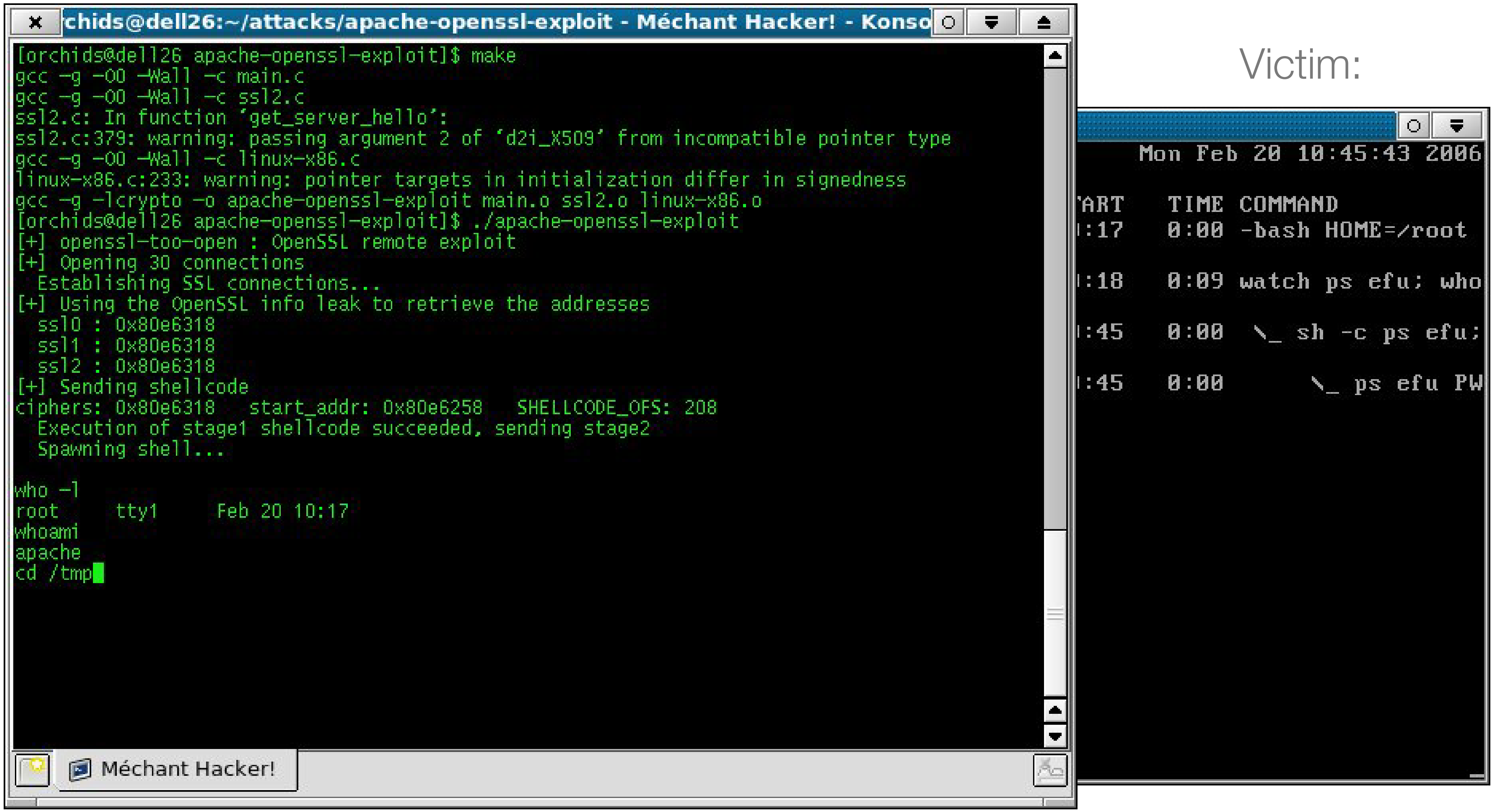


Works. Only `root` appears to be here (I am invisible...)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

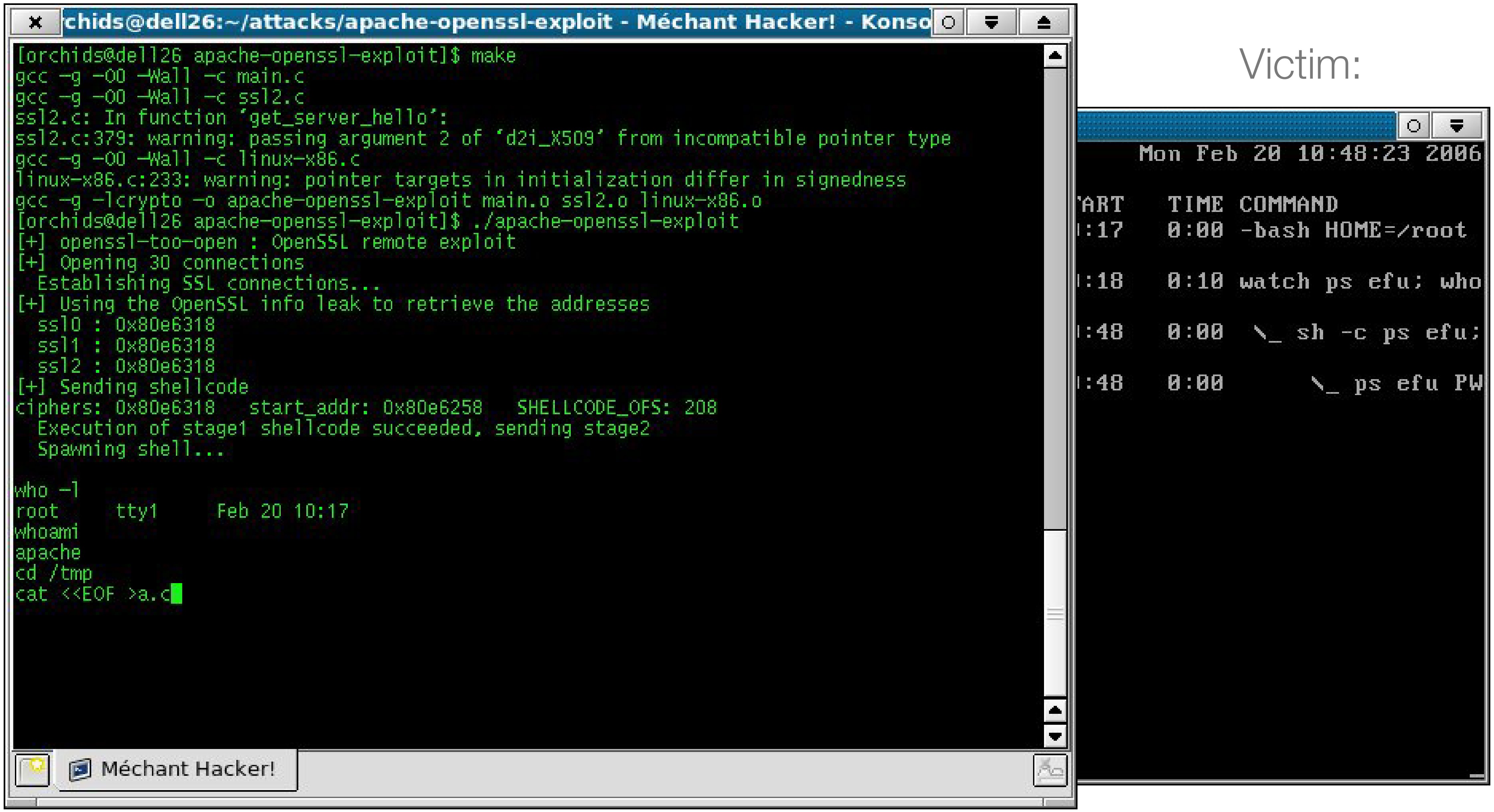


Next step: privilege escalation.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

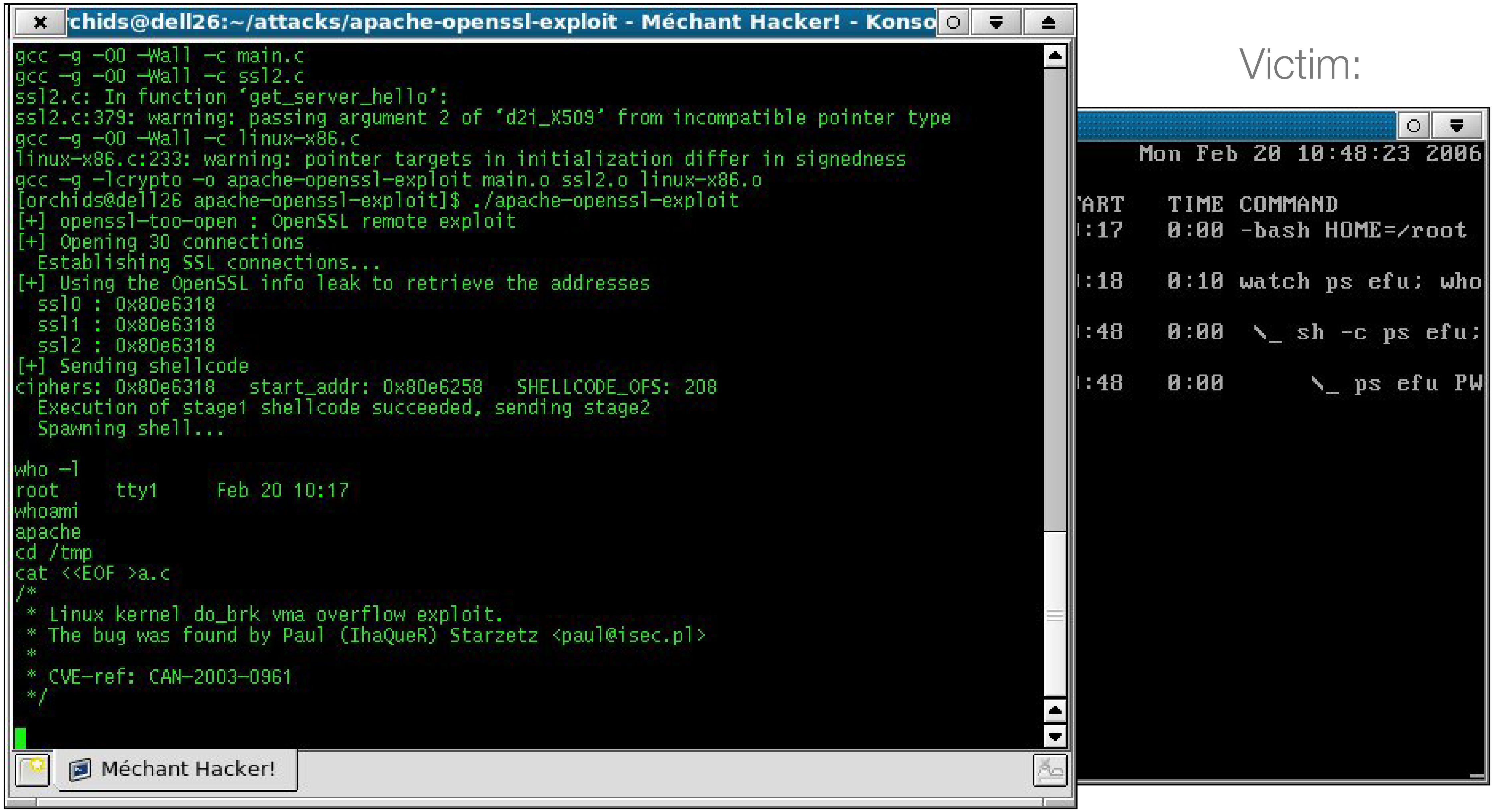


Next step: privilege escalation.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:



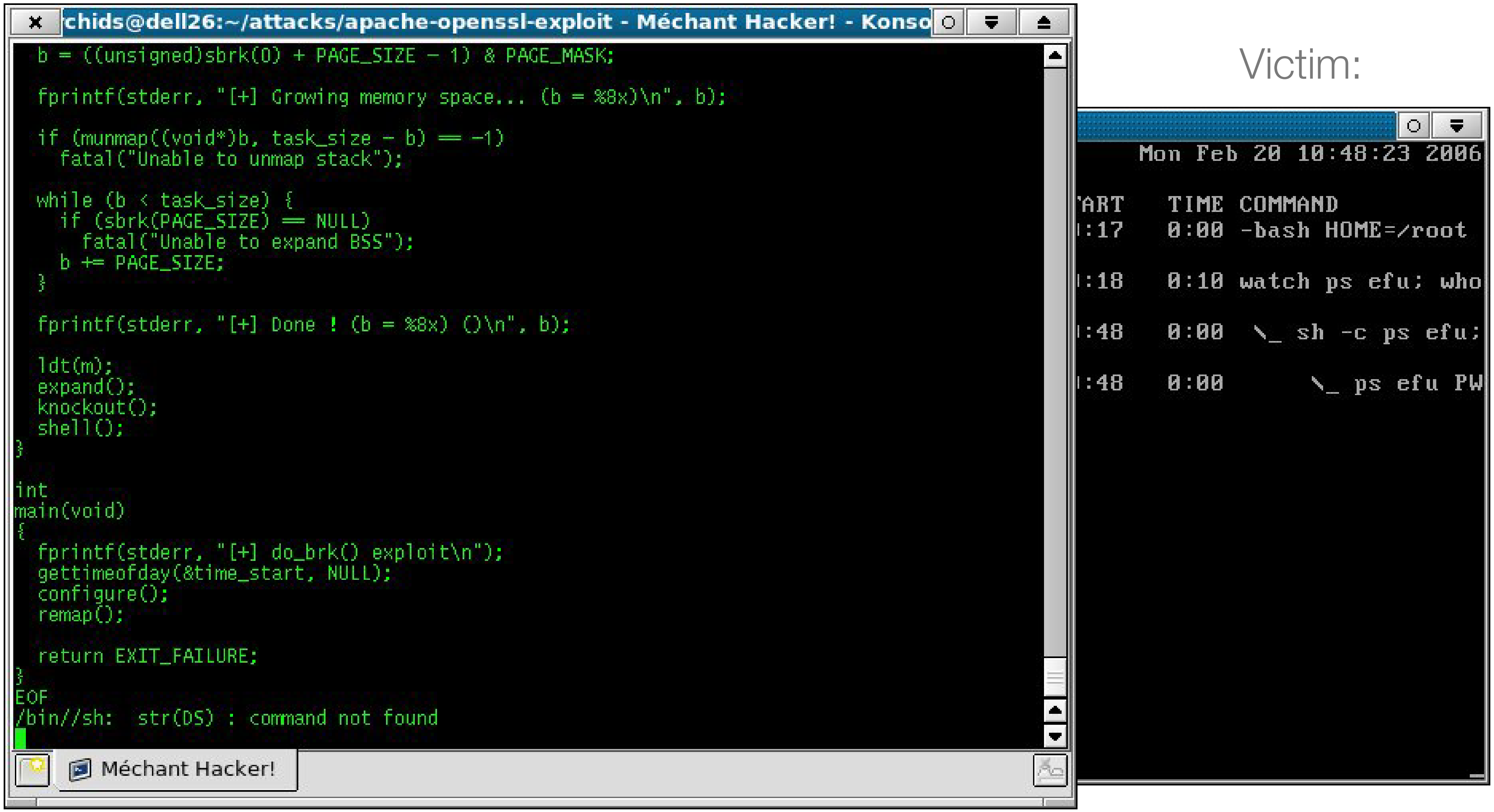
Next step: privilege escalation.

Let's use the do_brk attack for a change (Morton, Starzetz 2003)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:



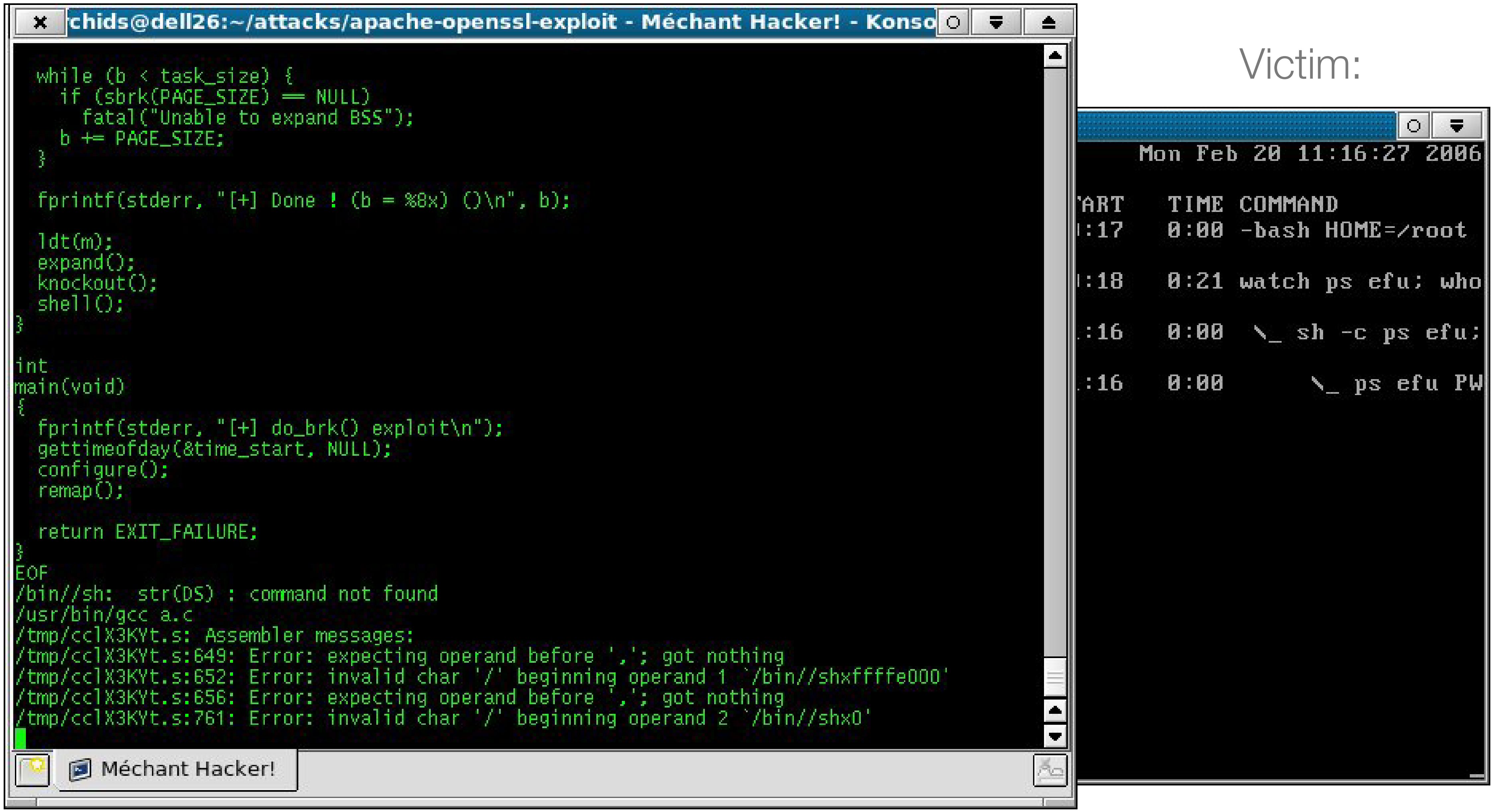
Next step: privilege escalation.

Let's use the do_brk attack for a change (Morton, Starzetz 2003)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:



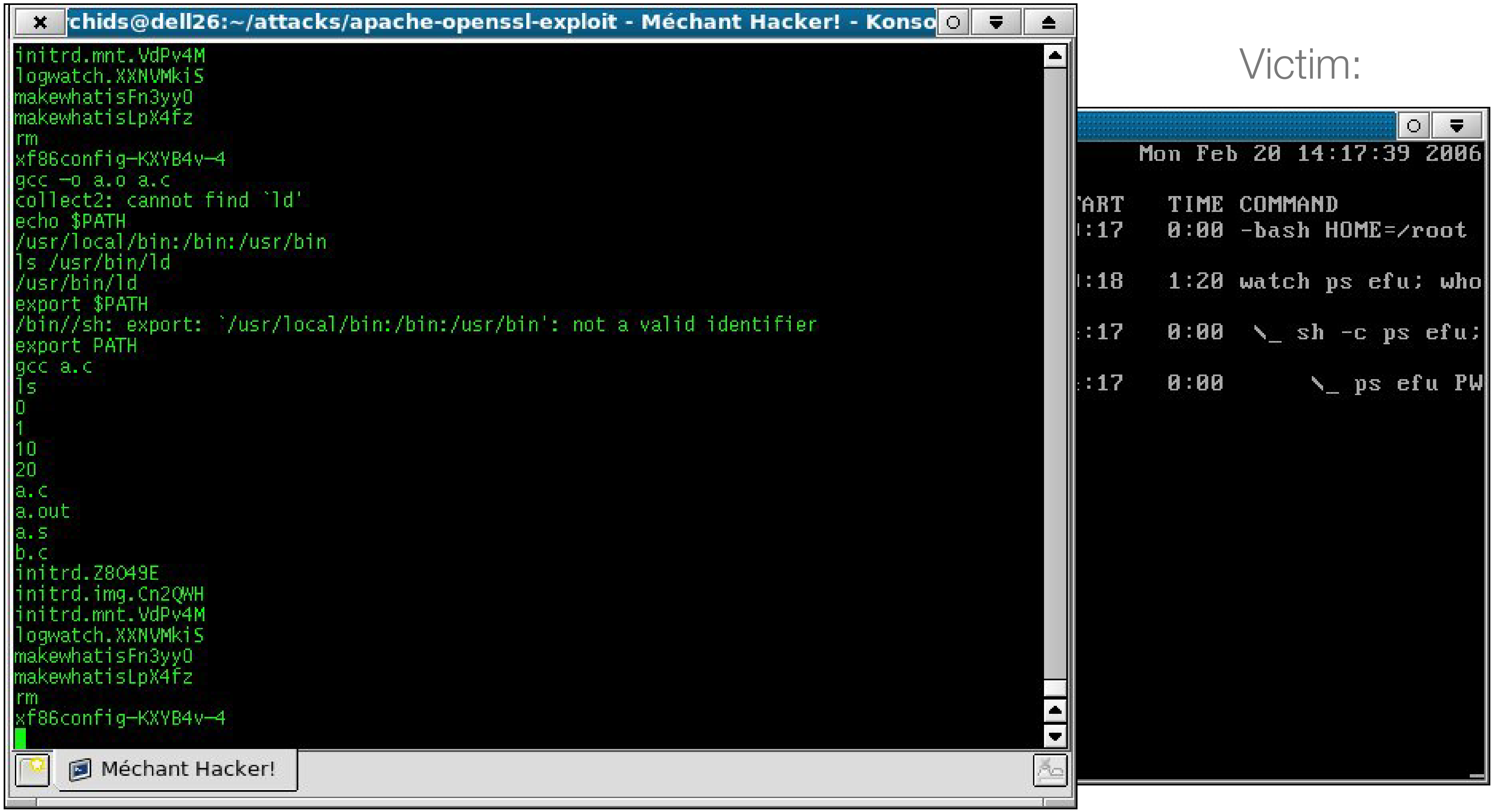
Next step: privilege escalation.

Let's use the do_brk attack for a change (Morton, Starzetz 2003)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:



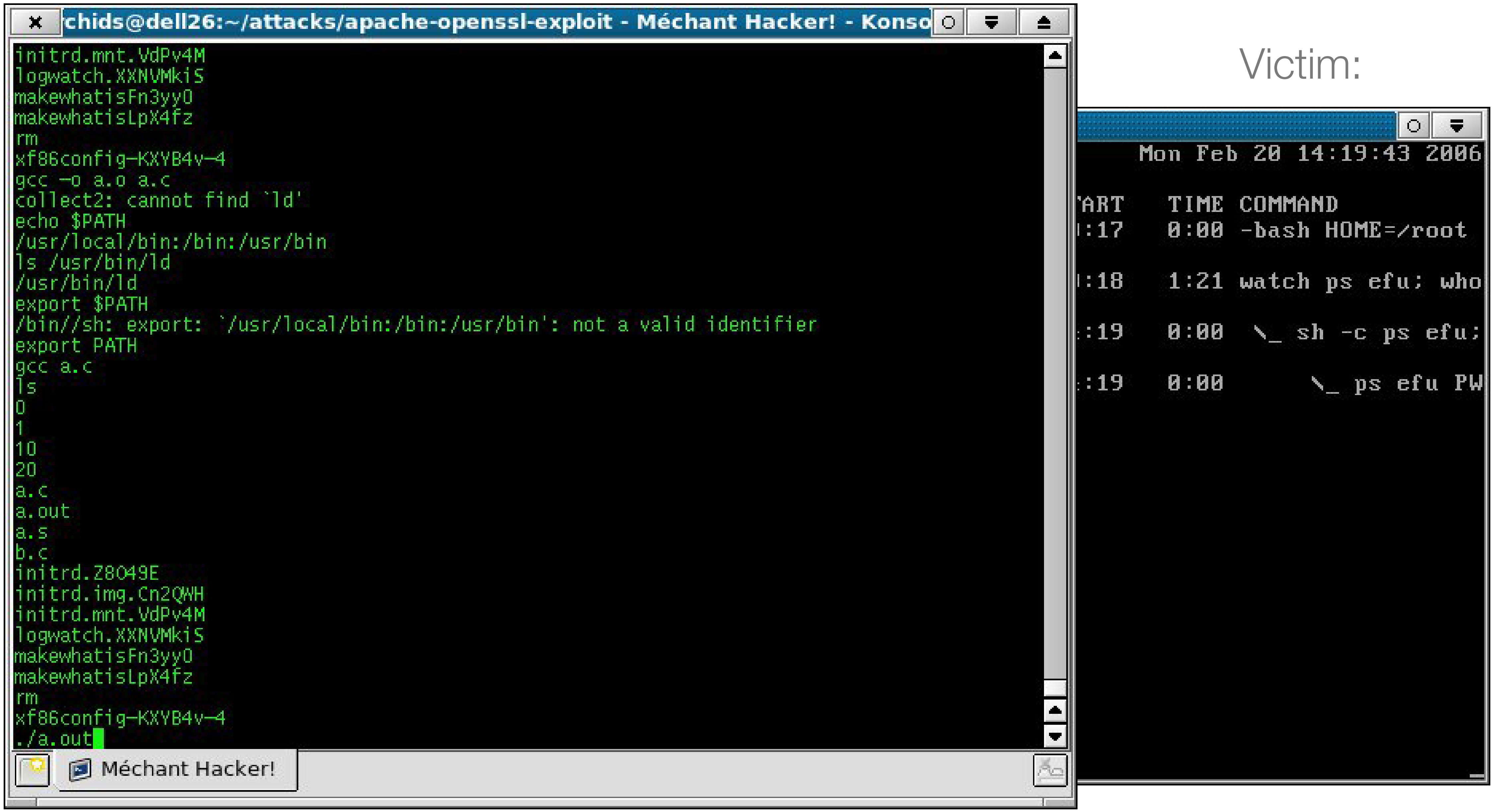
Next step: privilege escalation.

Let's use the do_brk attack for a change (Morton, Starzetz 2003)

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

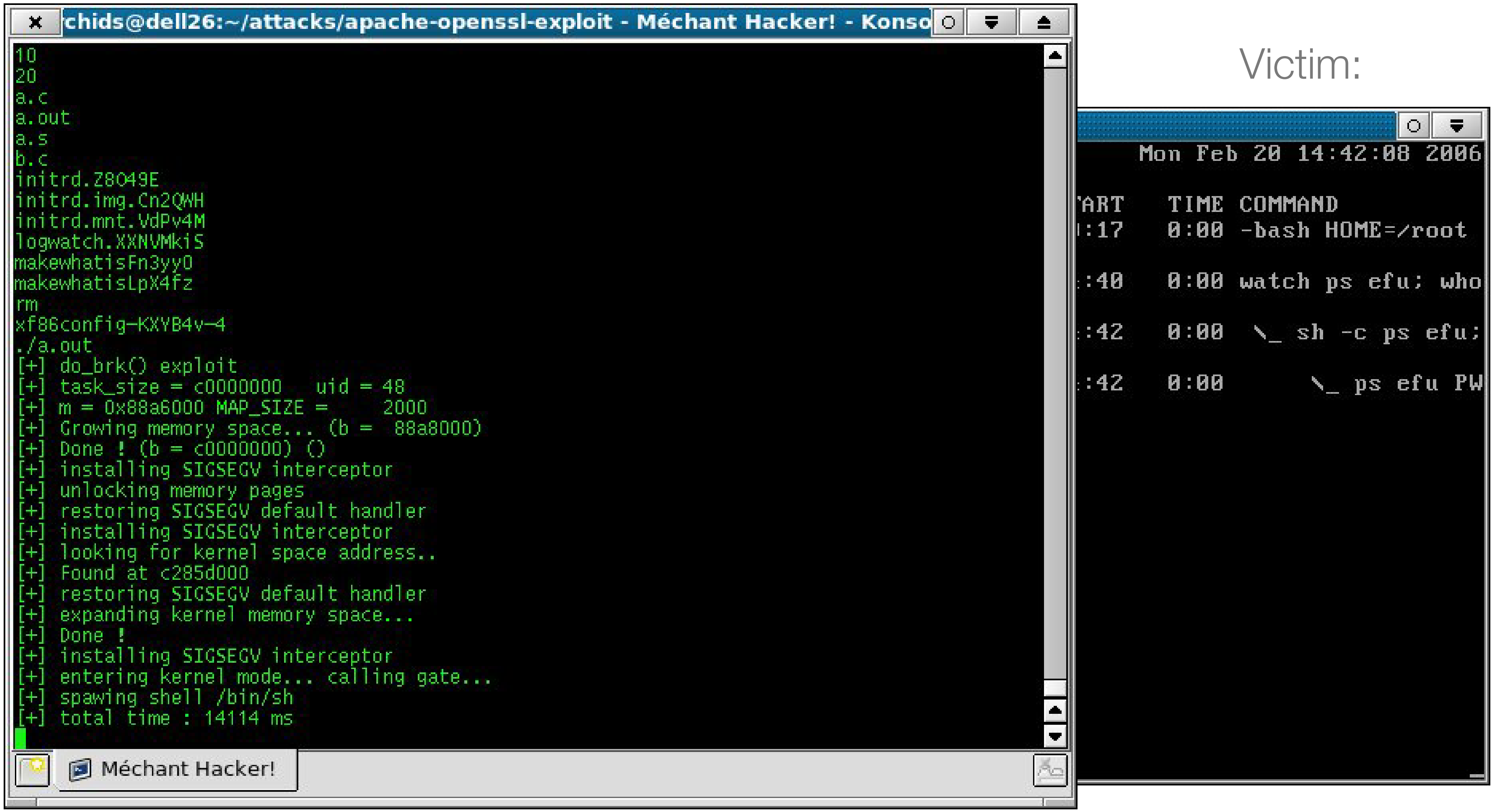


Here we are at last. Launch attack.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

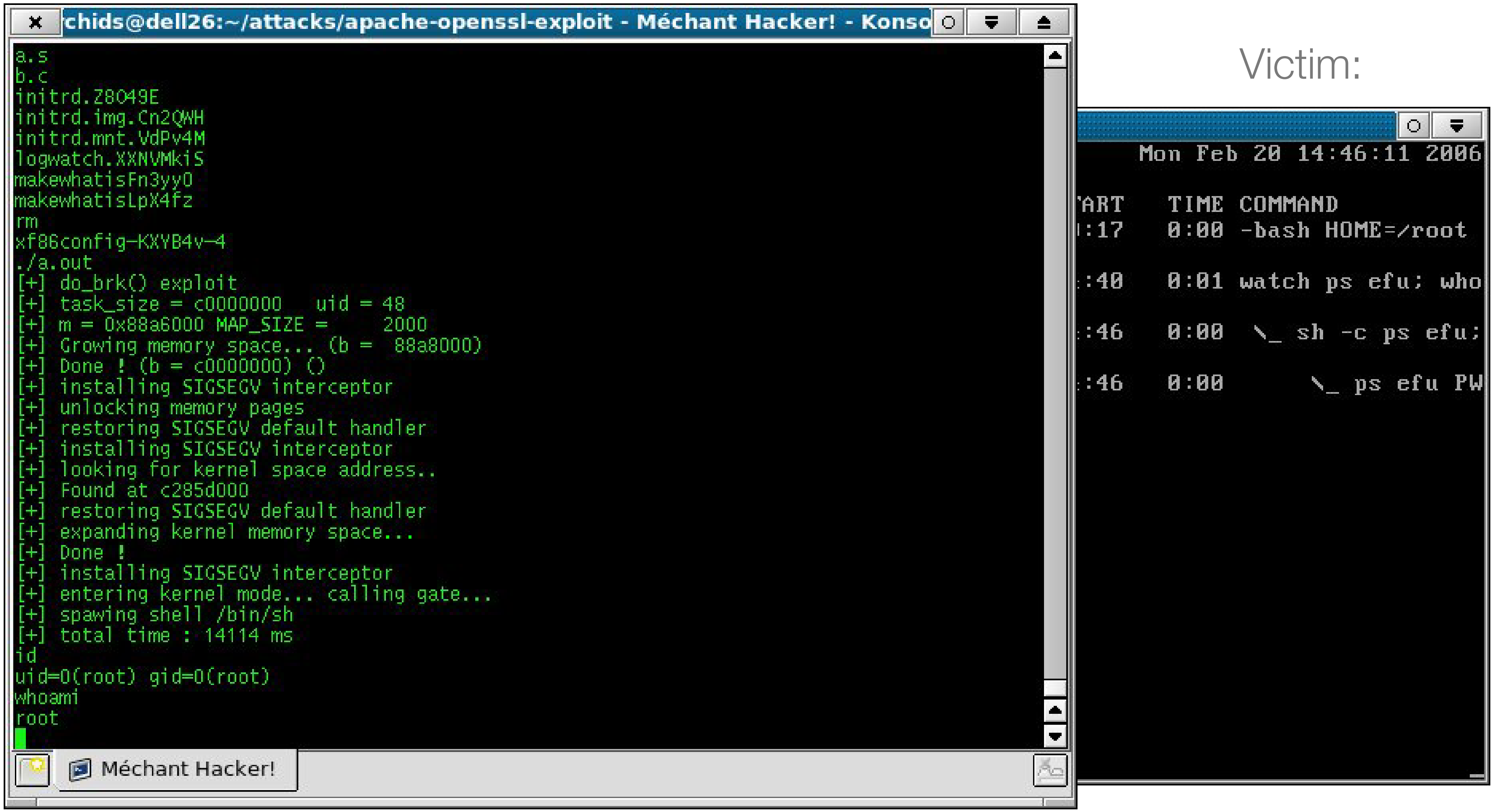


Works. I should have `root` privileges now.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

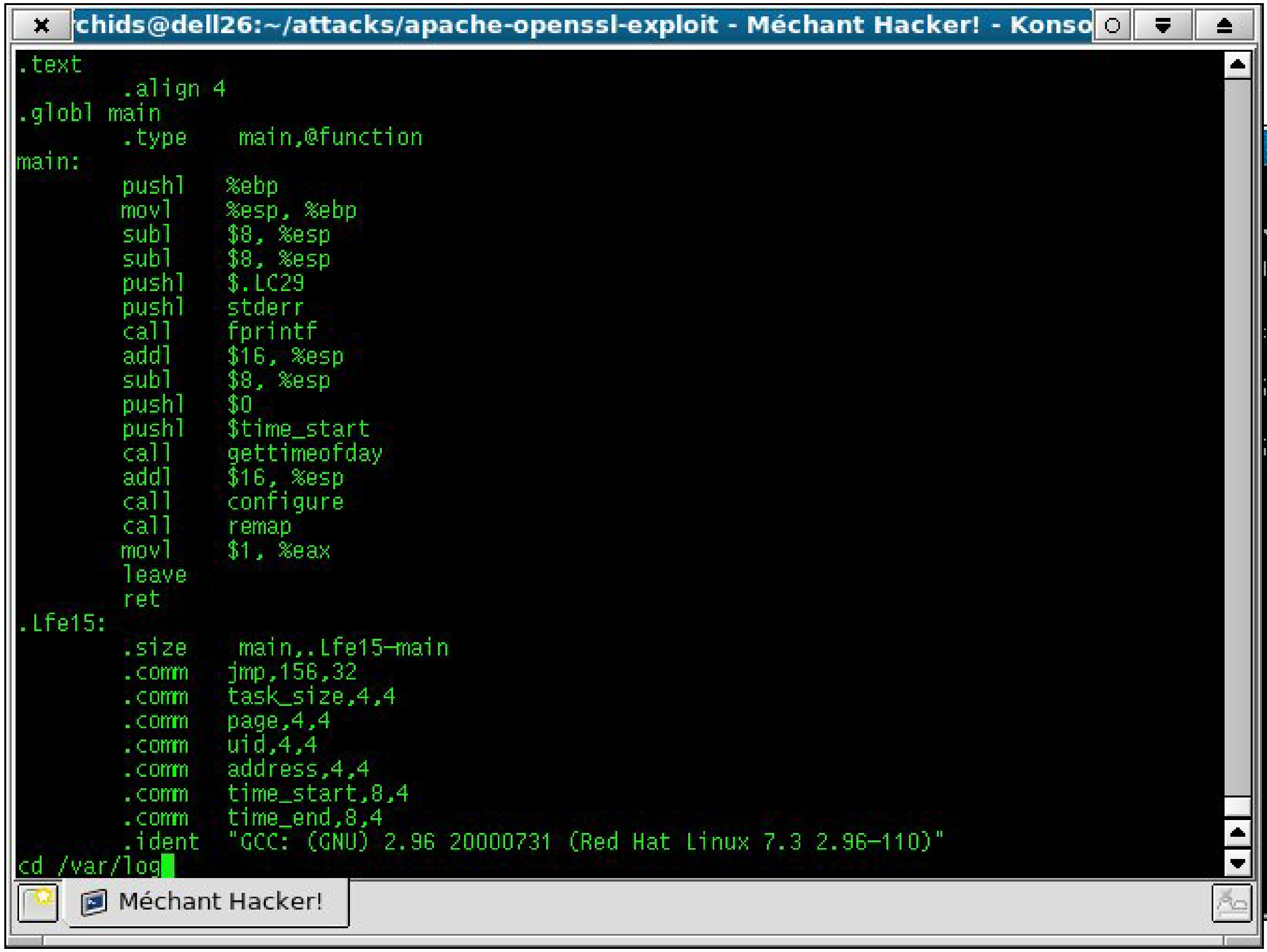
Victim:



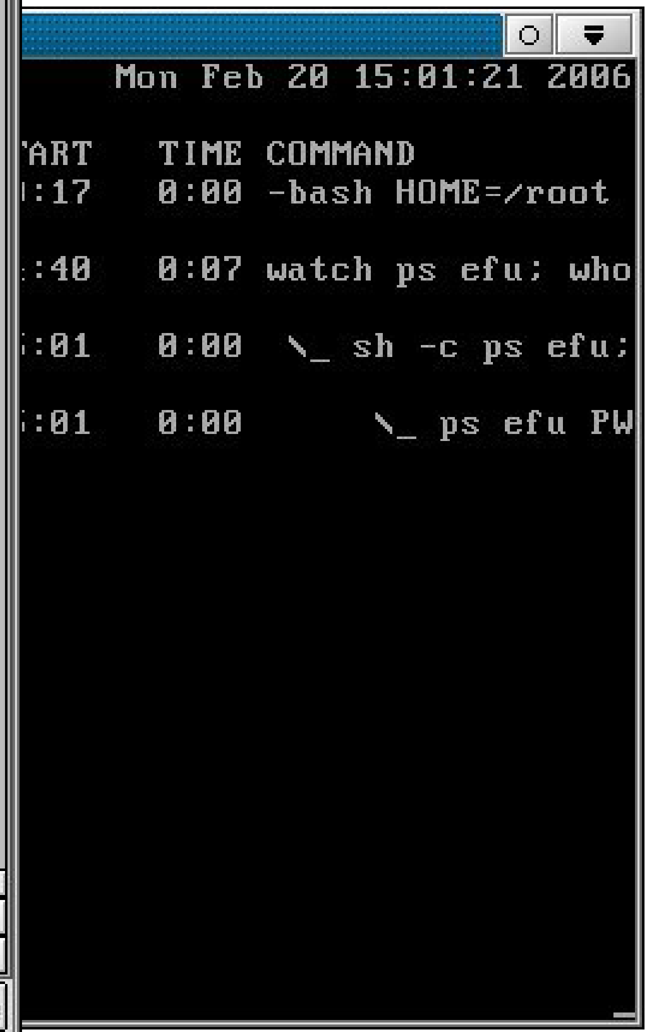
Works. I have `root` privileges.

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:



Victim:

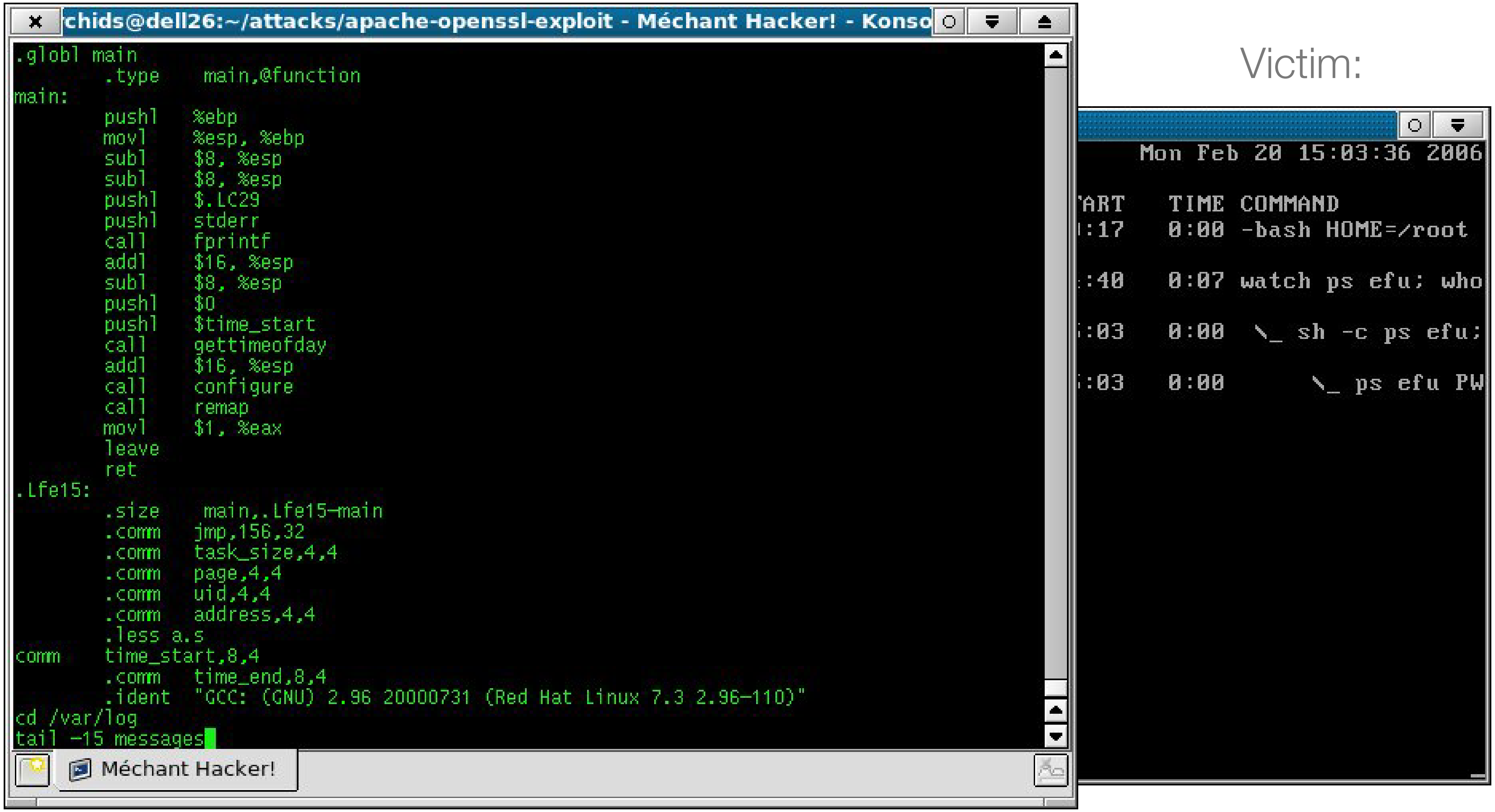


Check my tracks...

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

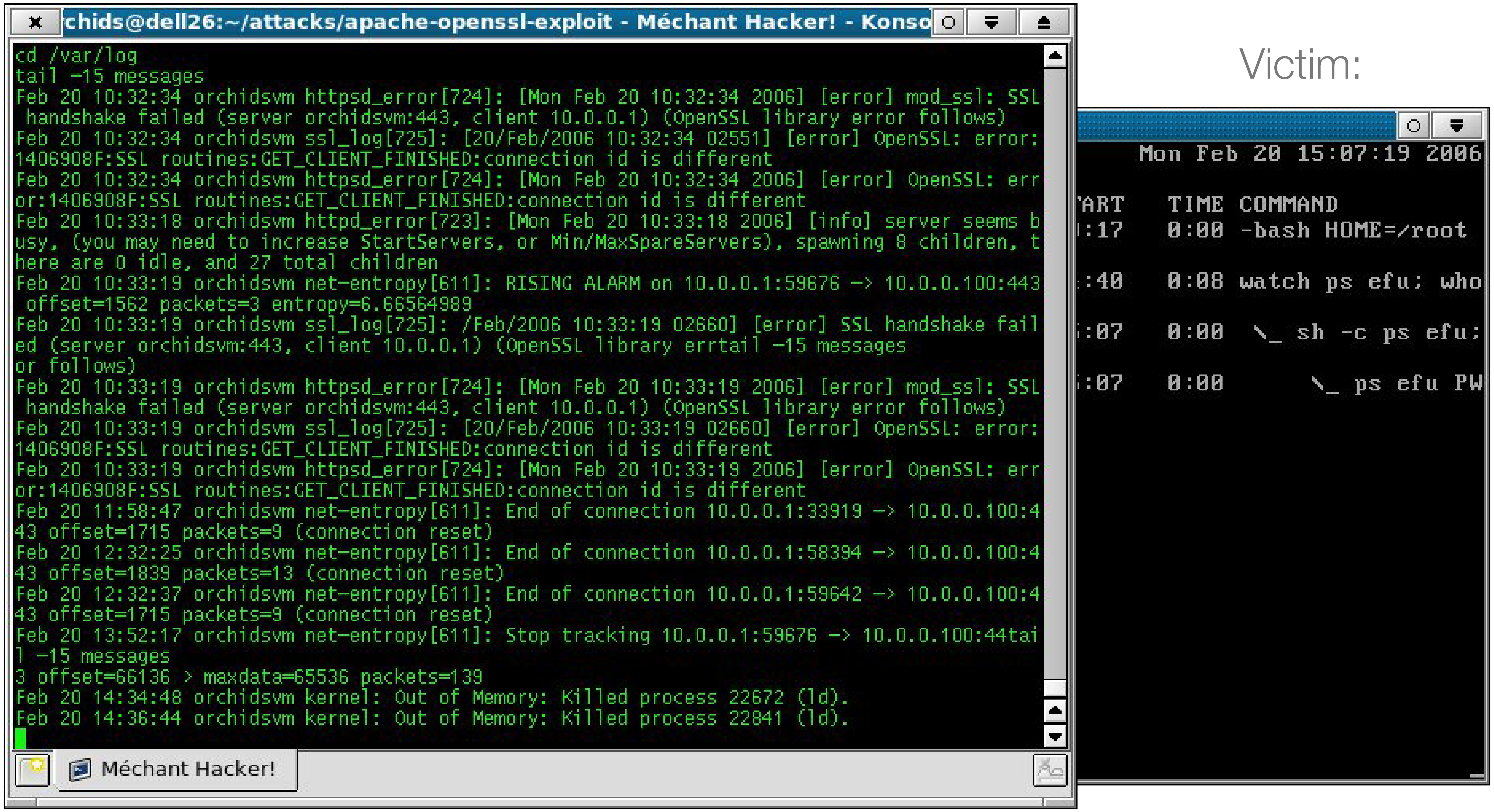


Check my tracks...

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:

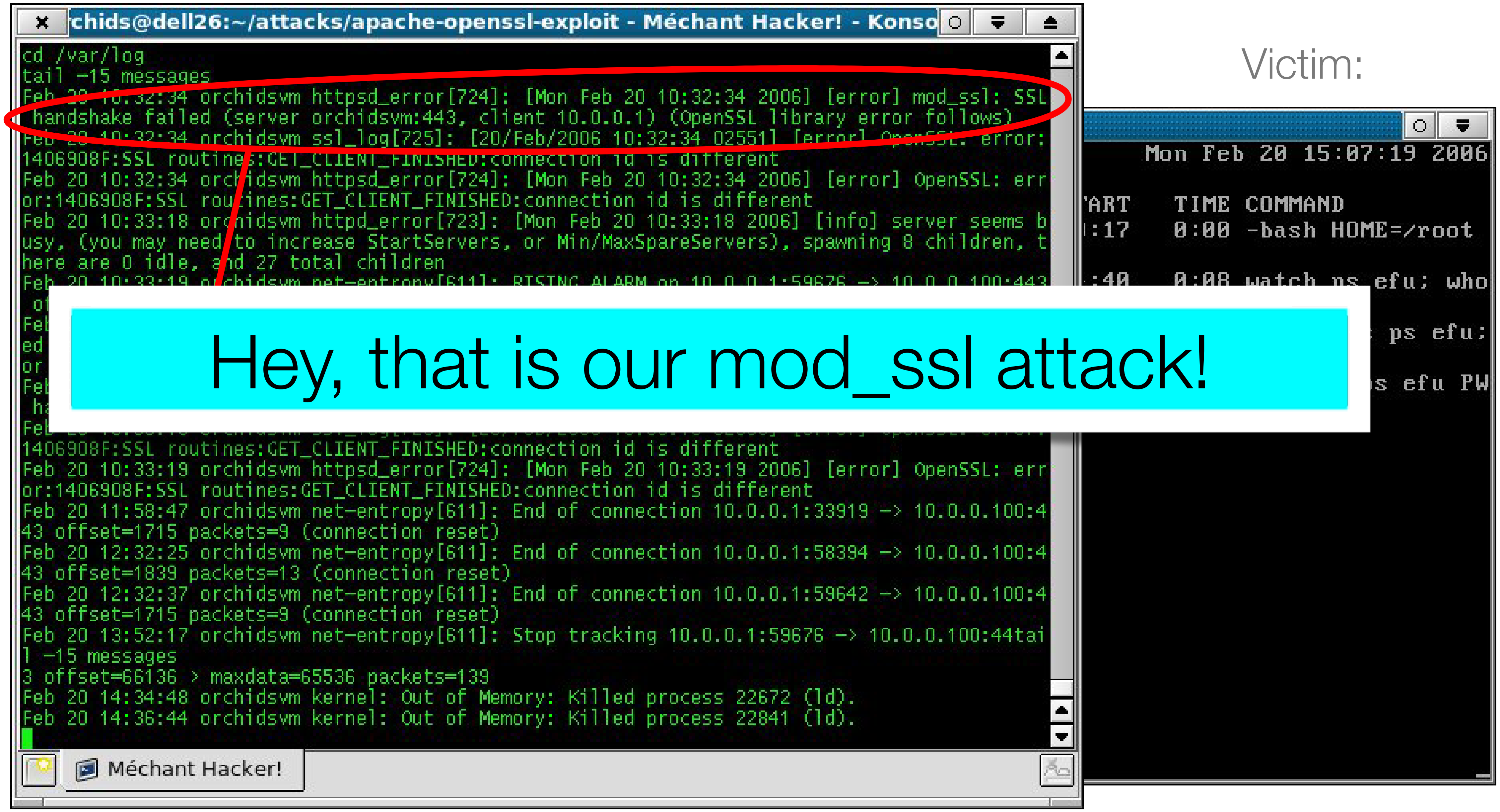


Check my tracks...

The mod_ssl remote-to-local attack (McDonald 2003)

Remote attacker:

Victim:



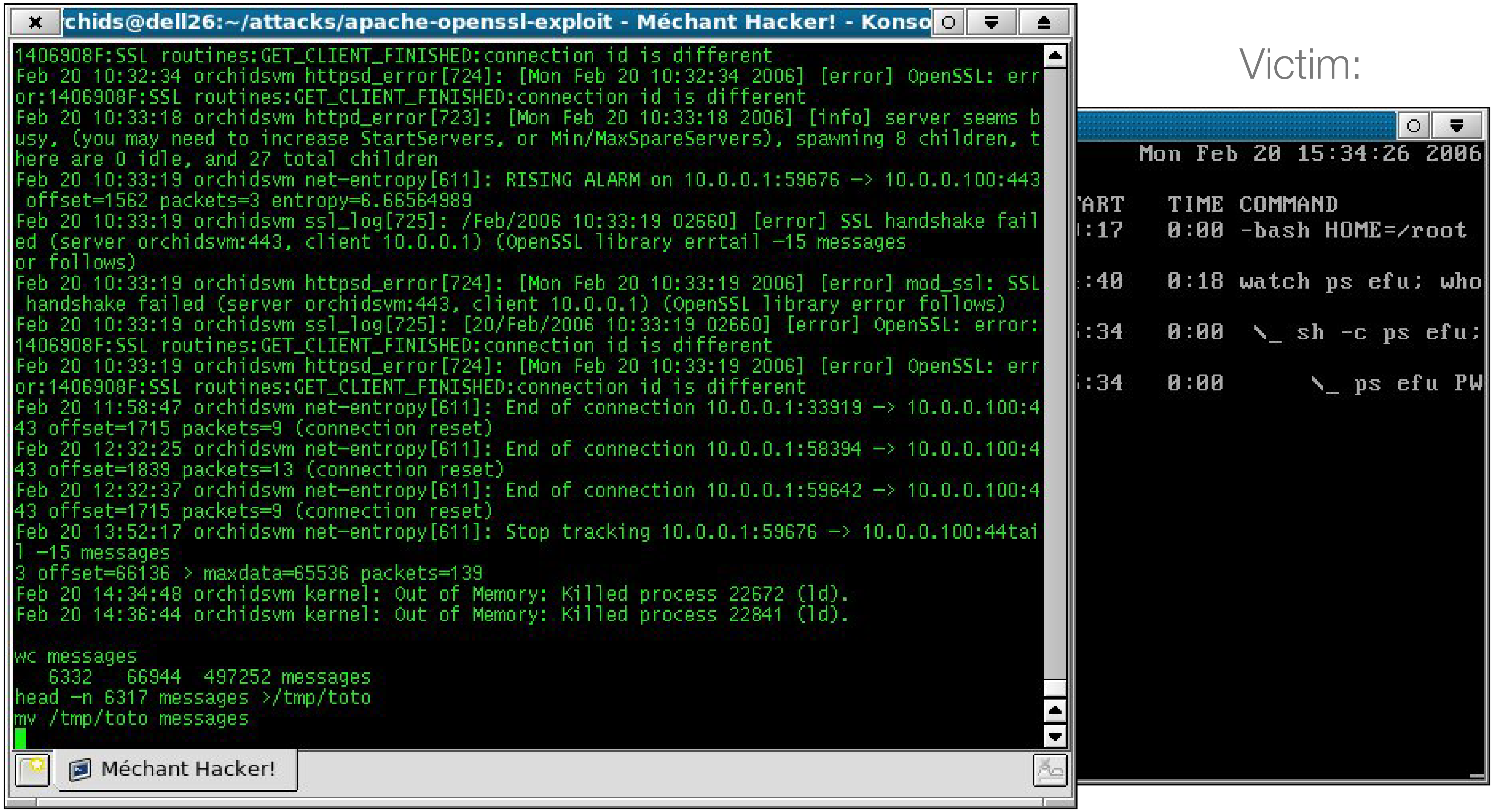
Hey, that is our mod_ssl attack!

Check my tracks... indeed mod_ssl attack causes SSL handshake to fail...

The mod_ssl remote-to-local attack (McDonald 2003)

Victim:

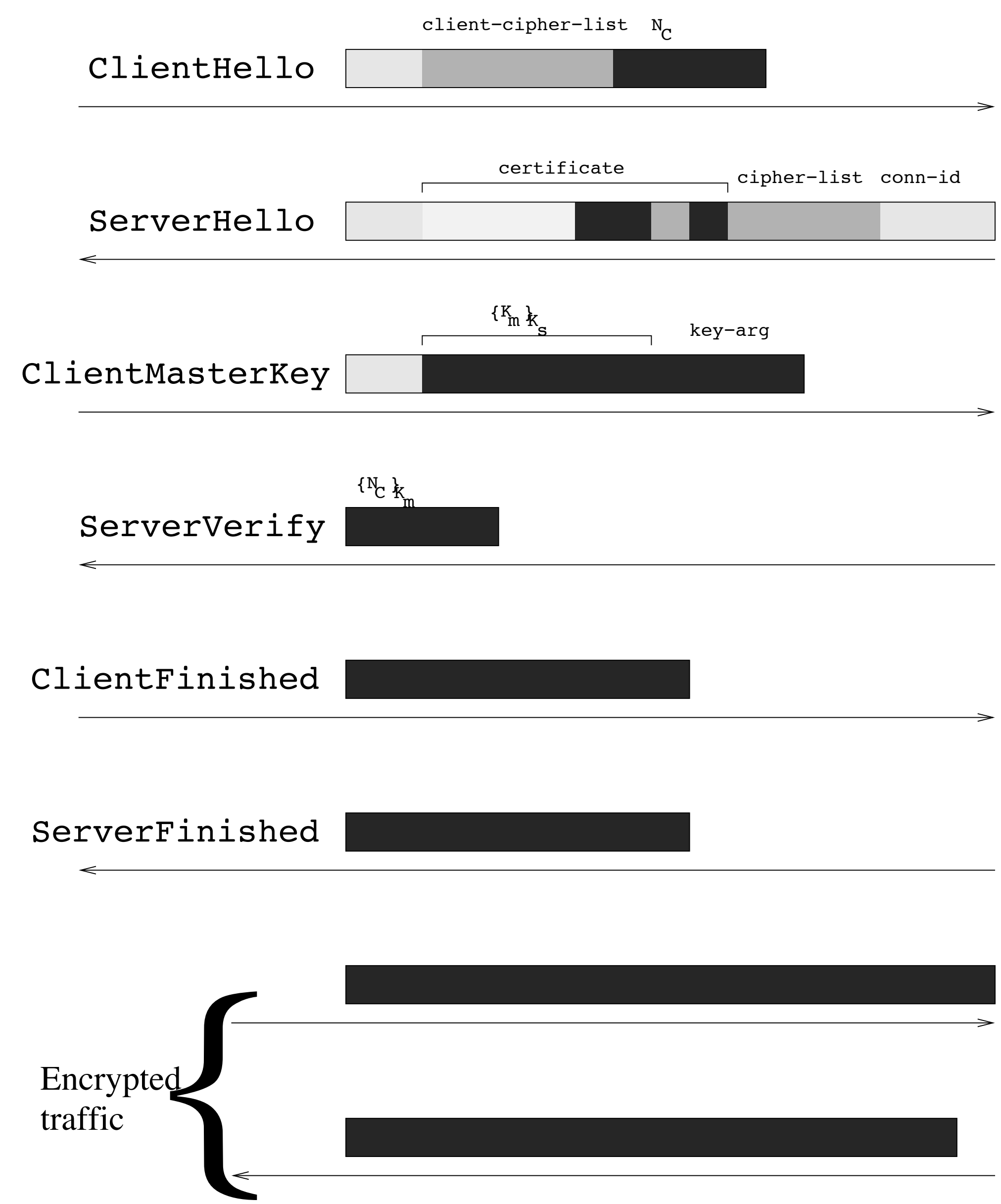
Remote attacker:



Check my tracks... OK, erase all compromising data.

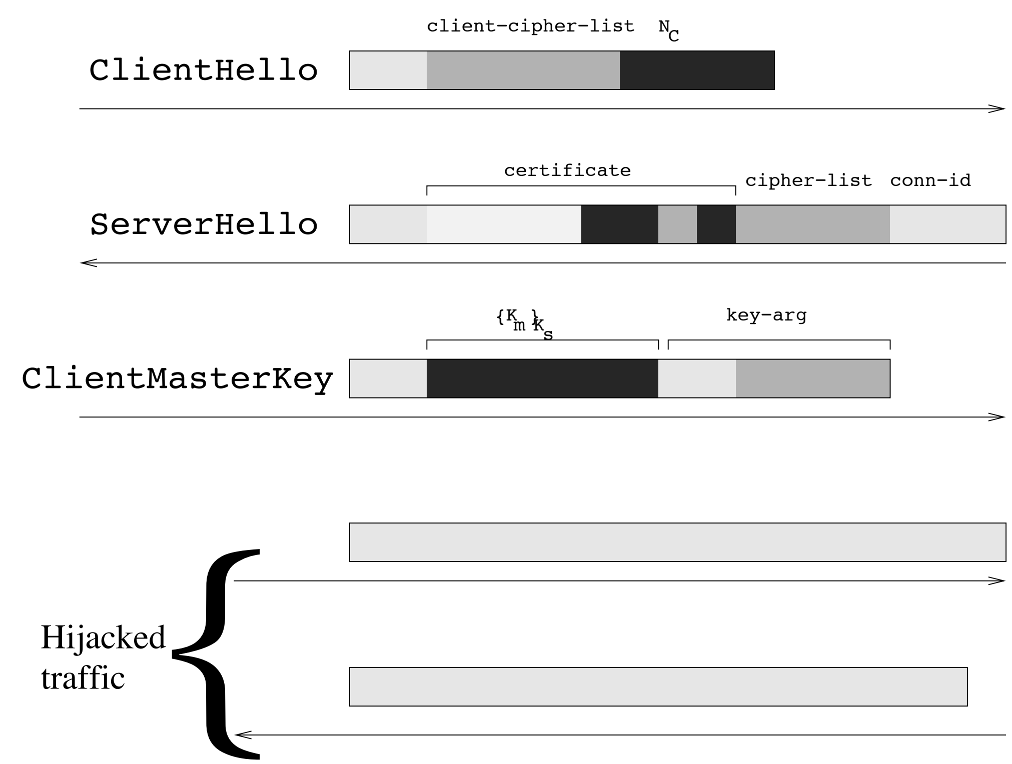
The mod_ssl remote-to-local attack (McDonald 2003)

- Normal SSL v2 handshake:
- **Black** zones are:
  - **random keys**/data
  - **encrypted** text
- Mod_ssl attack causes a **buffer overflow** on `key-arg`, allowing attacker to transmit useful info over the network, by abusing `free`().



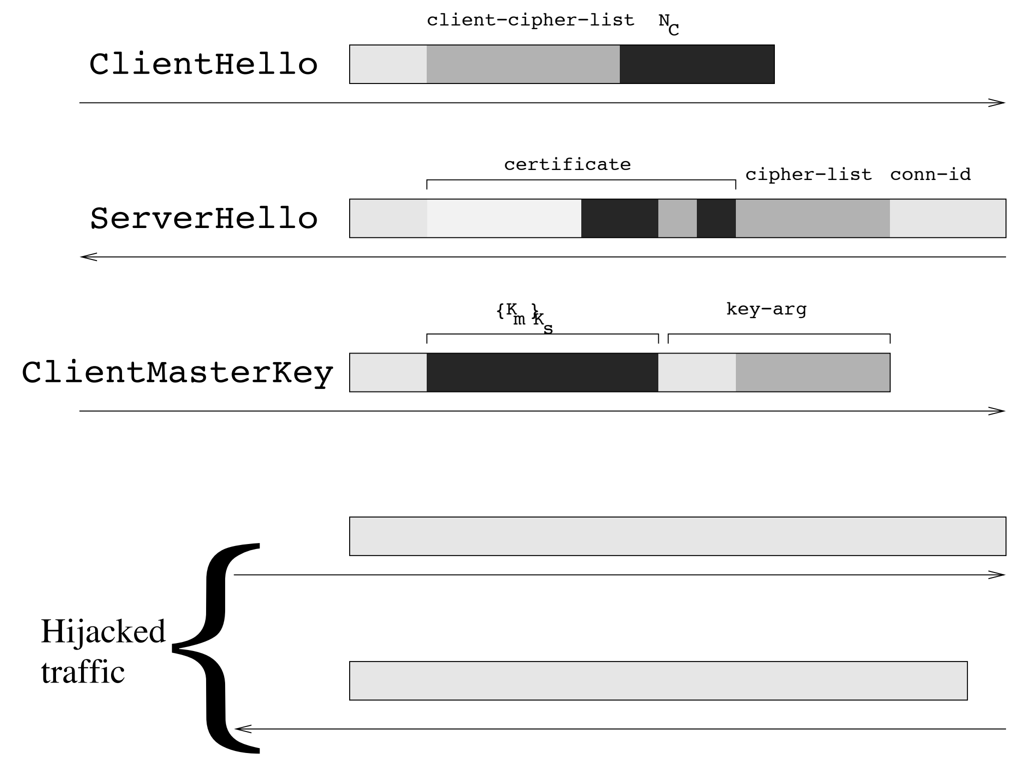
The mod_ssl remote-to-local attack (McDonald 2003)

- Hijacked SSL v2 handshake:
- **Black** zones are:
  - **random keys**/data
  - **encrypted** text
- Note that key-arg is now «**less random-looking**».
- Subsequent traffic no longer looks random either.



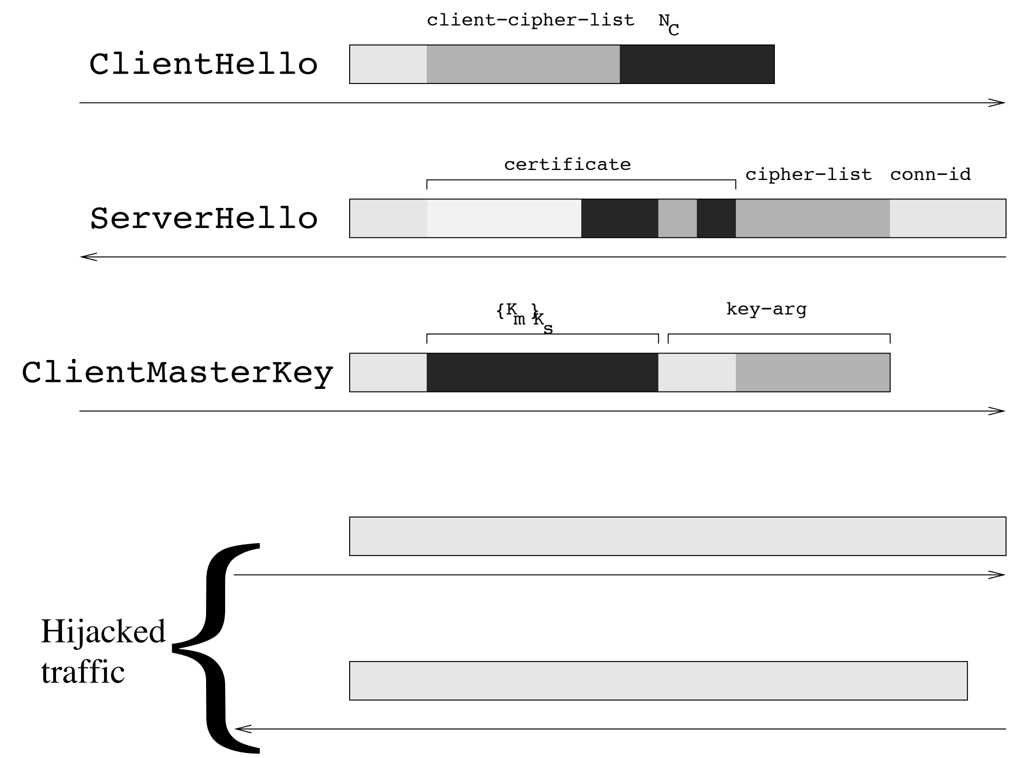
The mod_ssl remote-to-local attack (McDonald 2003)

- Hijacked SSL v2 handshake:
- **Black** zones are:
  - **random keys**/data
  - **encrypted** text
- Note that key-arg
  is now «**less random-looking**».
- Subsequent traffic no longer looks
  random either.



The mod_ssl remote-to-local attack (McDonald 2003)

- Hijacked SSL v2 handshake:
- **Black** zones are:
  - **random keys**/data
  - **encrypted** text
- Note that key-arg
  is now «**less random-looking**».
- Subsequent traffic no longer looks
  random either.



NetEntropy: a tool to compute **statistical entropy** on-line and compare them against a profile of normal behavior

Related work

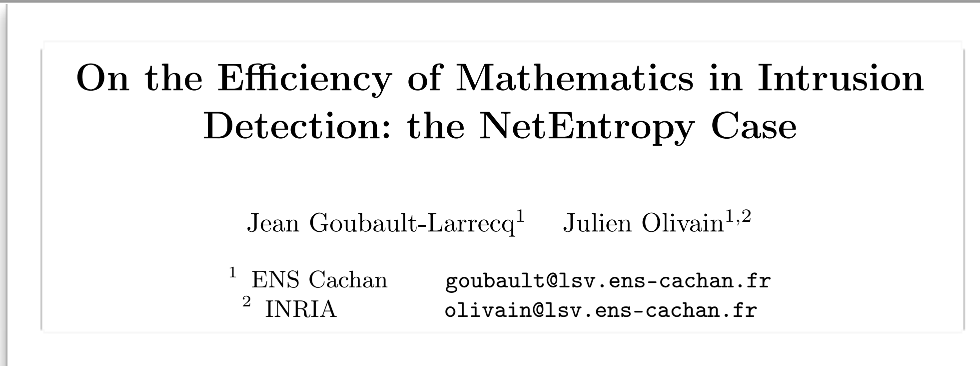
- **Shannon** (1948): theory of communication
  «random-looking» = entropy H should be about 8 bits/byte **in the limit**
  ... but we should react as soon as we can (**fewer** bytes)

- **Entropy** computation part of:
  packer detector **PEiD**, file system forensic analysis tool **WinHex**, etc.

- Packet type classifier tool **PAYL** [Wang, Cretu, Stolfo 2005]
  uses Mahalanobis distance **clustering**

- Our problem is simpler: is payload **random-looking**?

NetEntropy: entropy-based classification


On the Efficiency of Mathematics in Intrusion Detection: the NetEntropy Case

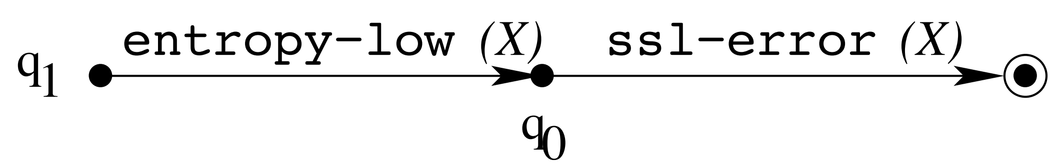
Jean Goubault-Larrecq[1] Julien Olivain[1,2]

[1] ENS Cachan goubault@lsv.ens-cachan.fr
[2] INRIA olivain@lsv.ens-cachan.fr

In *FPS*'13, Springer Verlag LNCS, 2014.

http://www.lsv.ens-cachan.fr/net-entropy/

- Still being downloaded 1-2 times a week
- Incorporated as an ORCHIDS module, but can be used as a standalone tool



- One of our **best**-cited papers, e.g.:
  [Lyda, Hamrock 2007]
  [Dorfinger, Panholzer, Trammel, Pepe 2010]
  [Dorfinger, Panholzer, John 2011]
  [Han Zhang, Papadopoulos, Massey 2013]
  [Rossow, Dietrich 2013]

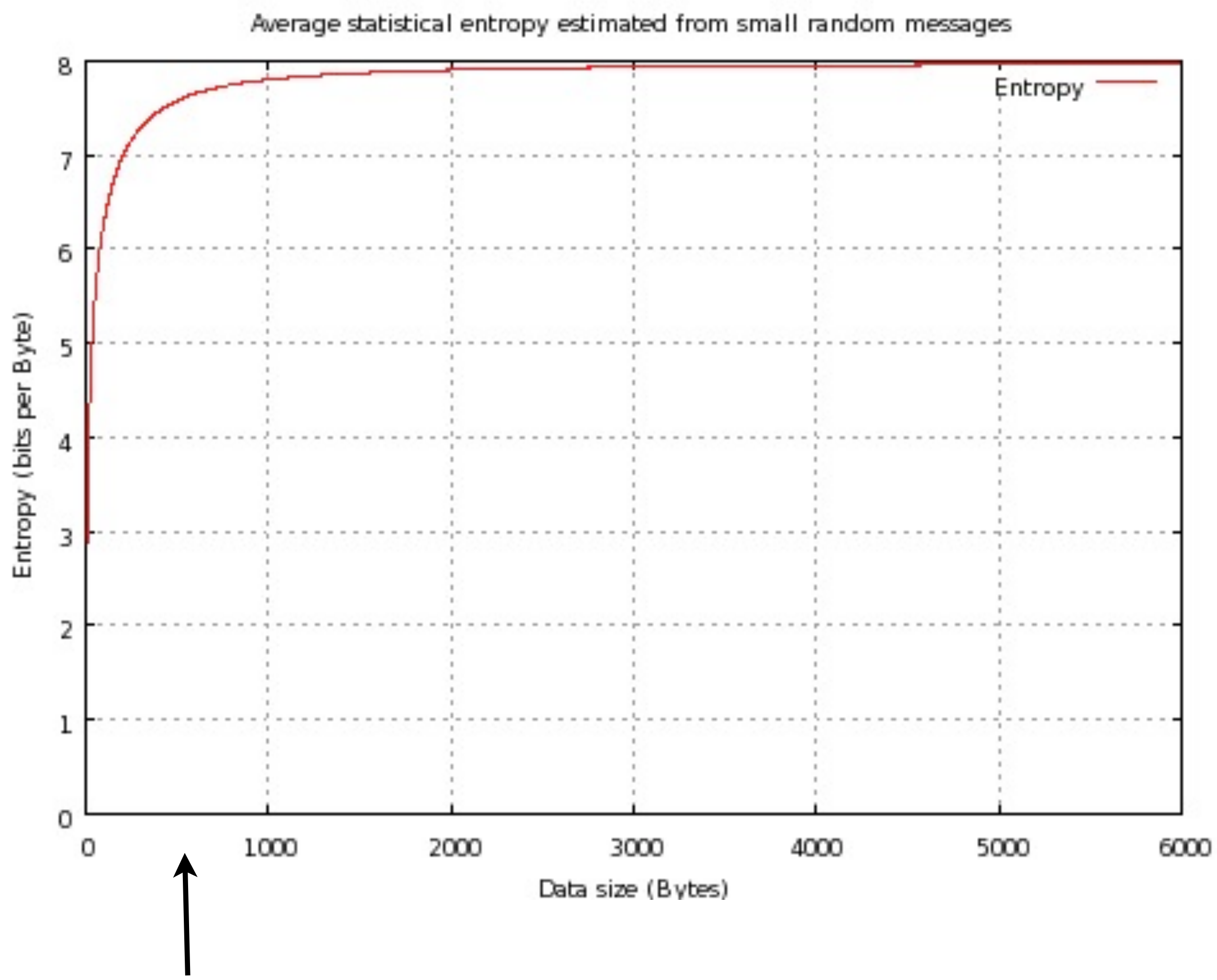
  ... mostly for detecting packers, Skype traffic, bots, etc.

NetEntropy: entropy-based classification

Two problems:

1. What should be statistical entropy like for **small** data sizes?
   («**undersampled**» case)

2. When should we decide that a flow is non-random?
   (how small are the **confidence intervals**?)

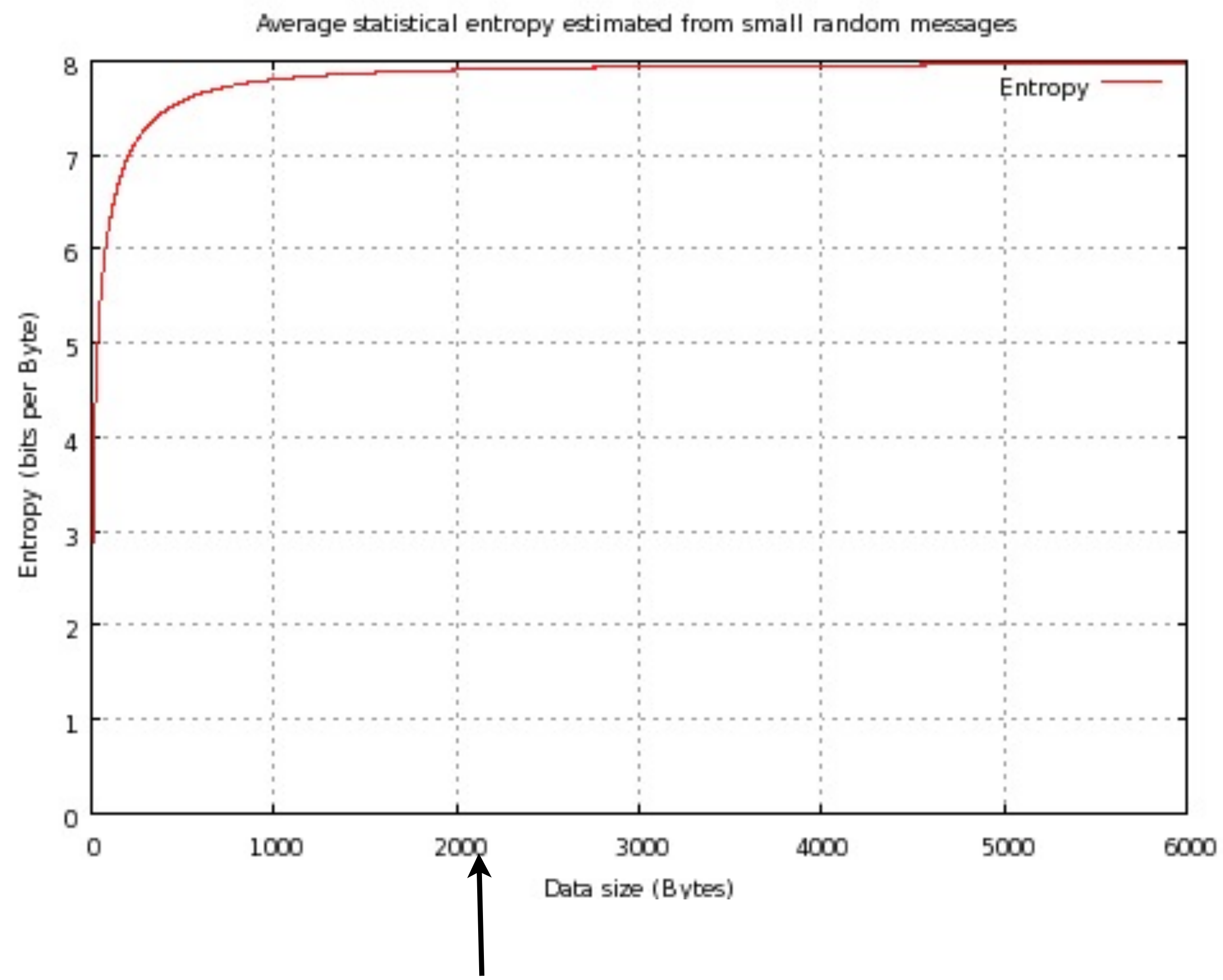


NetEntropy: entropy-based classification

Two problems:

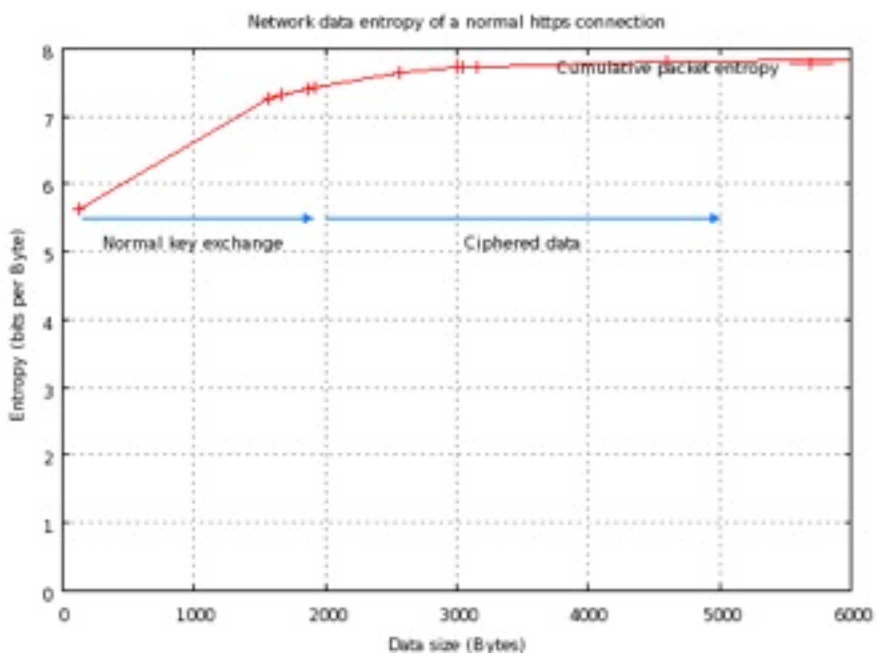
1. What should be statistical entropy like for **small** data sizes?
   («**undersampled**» case)

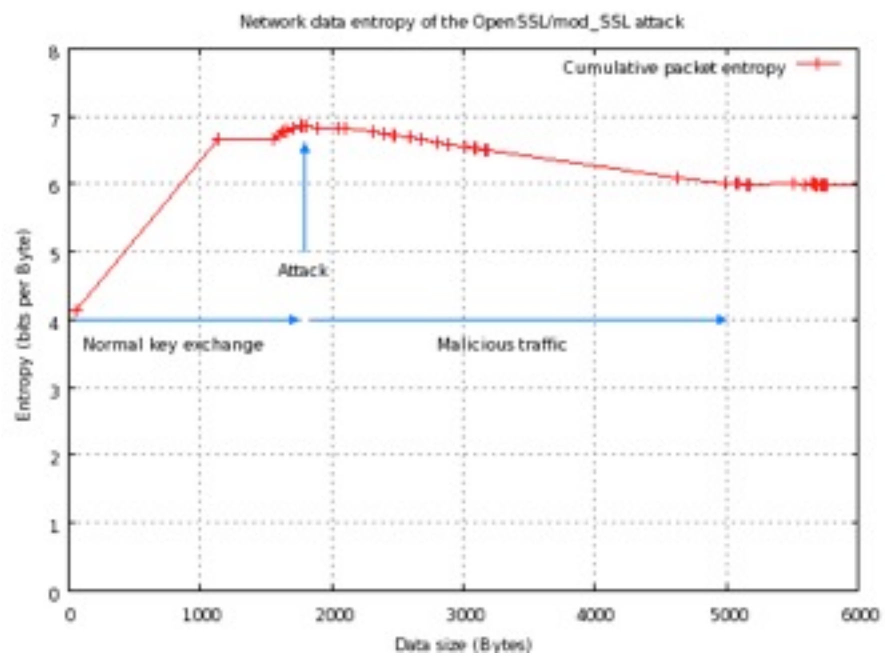
2. When should we decide that a flow is non-random?
   (how small are the **confidence intervals**?)

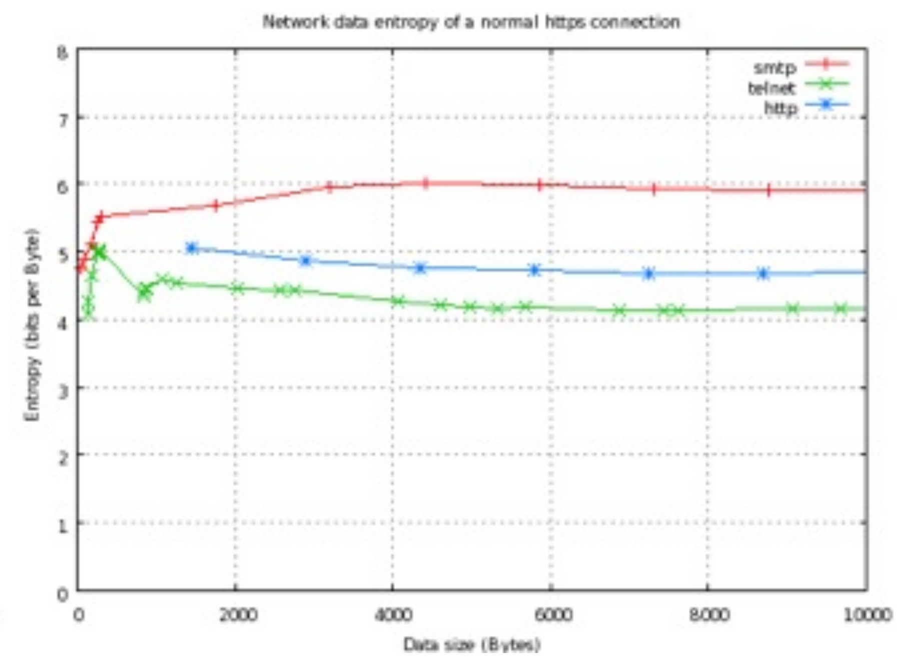


NetEntropy: entropy-based classification

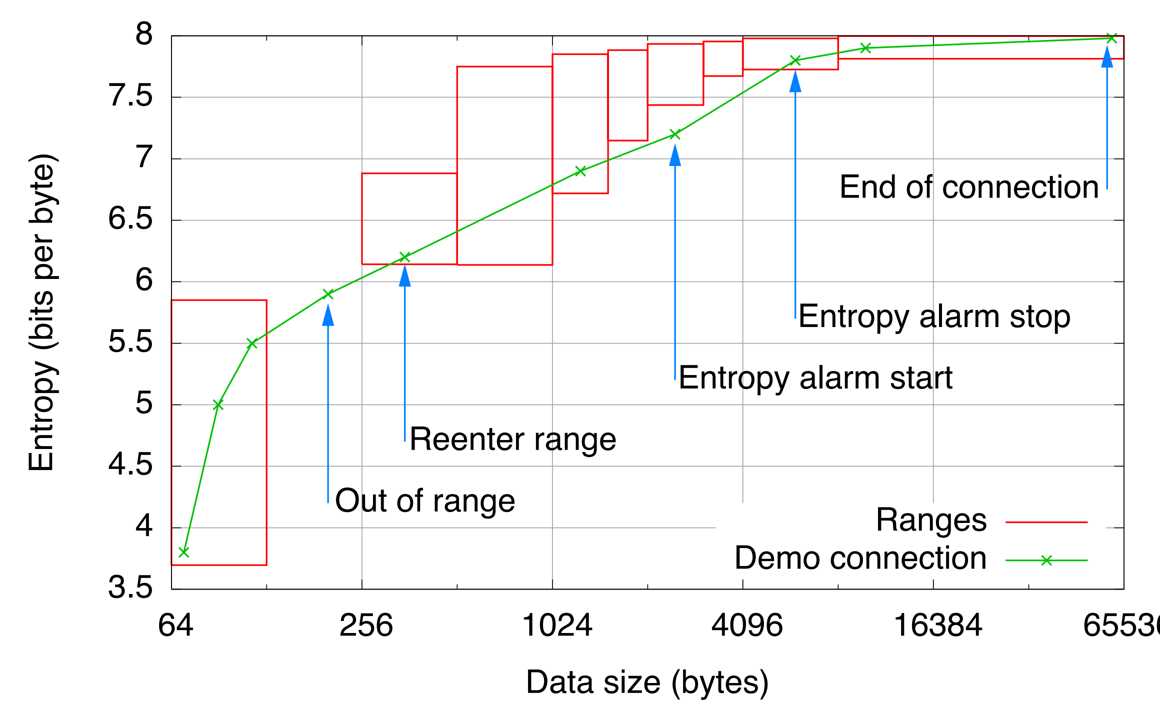
- In the end, we shall use **profile-based screening**, of course







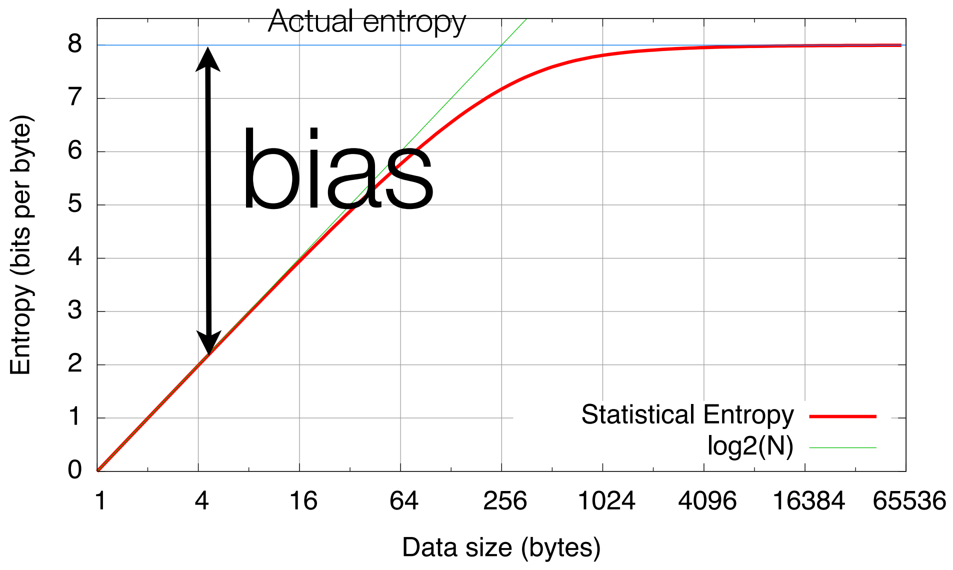
- But we do **science** to **understand why** it is working (and with which values)



Problem 1: good entropy estimators

Definition (statistical entropy):
For a flow of bytes w: $\hat{H}_N^{MLE}(w) = -\sum_{i=0}^{m-1} f_i \log f_i$
where f_i is frequency of letter i, $m = 256$

- How do you compute this?
- Change the problem: what is the **bias** between statistical and actual entropy?
- Several known estimators:
  [Miller, Madow 1955]
  «jackknifed» [Efron, Stein 1981]
  [Paninski 2004]



The Paninski estimator

Definition (Paninski):

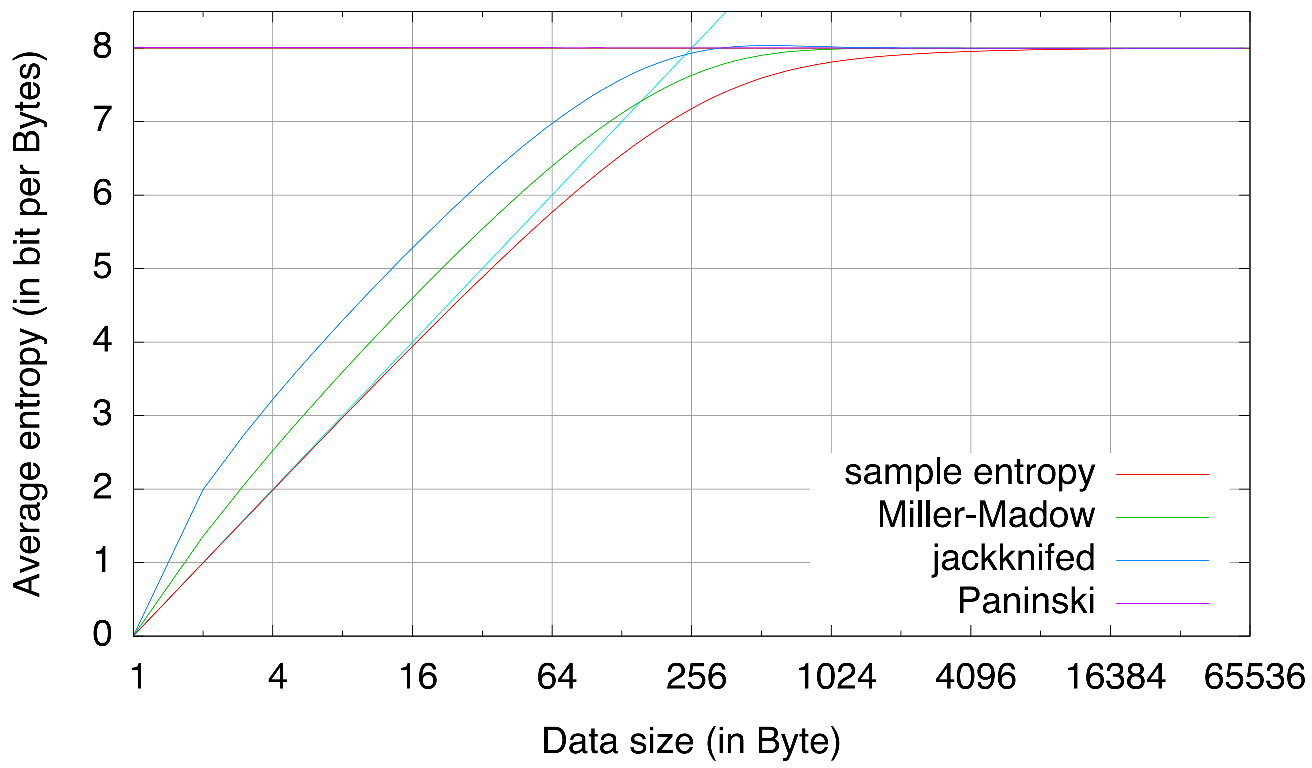
$$\hat{H}_N^P(w) = \hat{H}_N^{MLE}(w) - \log c + e^{-c} \sum_{j=1}^{+\infty} \frac{c^{j-1}}{(j-1)!} \log j$$

(m=256, c=N/m, N=#bytes read, uniform random source)

- Is meant to estimate the entropy of a **uniform**, **random source** as a correction to statistical entropy

- In our case, the closer the estimate to $H(w)$ = 8 the better

  Paninski looks perfect!



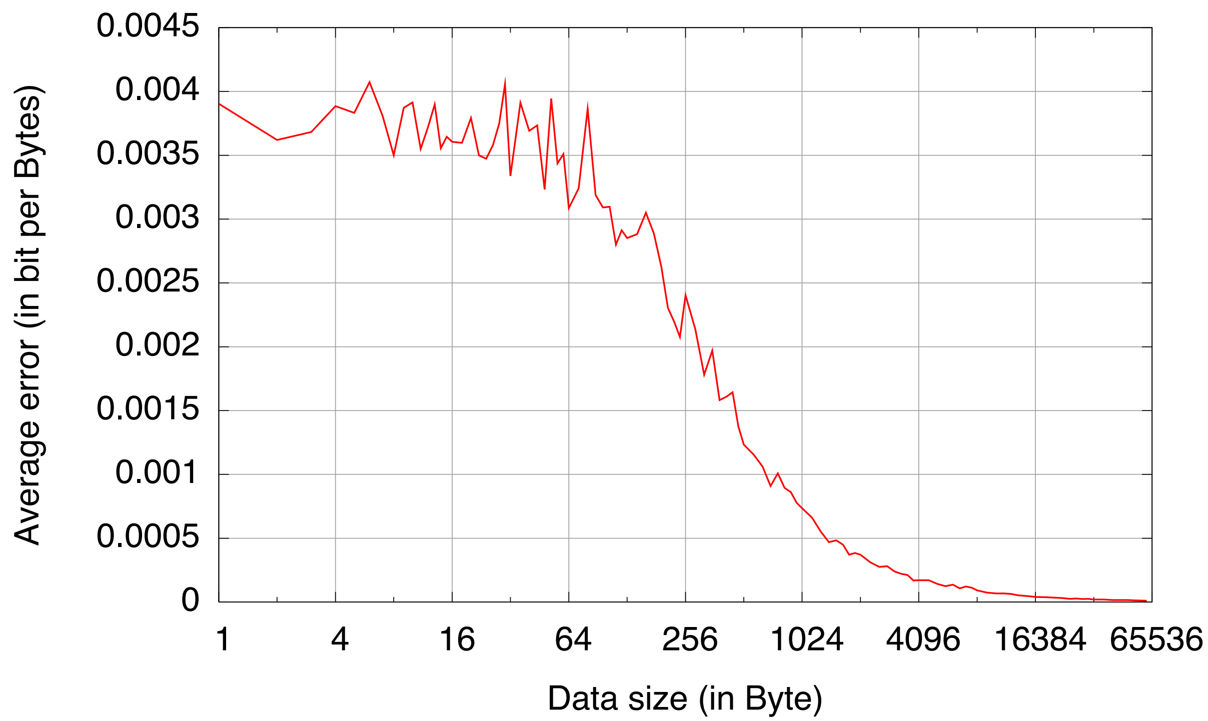
Problem 1 solved

- For N bytes read w, compare

  statistical entropy $\hat{H}_N^{MLE}(w)$

  with estimator $H_N(\mathcal{U}) = \log m + \log c - e^{-c} \sum_{j=1}^{+\infty} \frac{c^{j-1}}{(j-1)!} \log j + o(1)$

- **Extremely good** estimator!
- **Fast** to compute (tabulate anyway)
- The two quantities should be close iff w is **random-looking**
- (But how close? This is problem 2.)



Problem 2: confidence intervals

- Recognizing **text** as non-random: easy
- A bit more challenging:

```
0x55 0x89 0xe5 0x83 0xec 0x58 0x83 0xe4
0xf0 0xb8 0x00 0x00 0x00 0x00 0x29 0xc4
0xc7 0x45 0xf4 0x00 0x00 0x00 0x00 0x83
0xec 0x04 0xff 0x35 0x60 0x99 0x04 0x08
```

- **Is this random?**

Problem 2: confidence intervals

- Recognizing **text** as non-random: easy
- A bit more challenging:

0x55	0x89	0xe5	0x83	0xec	0x58	0x83	0xe4
0xf0	0xb8	0x00	0x00	0x00	0x00	0x29	0xc4
0xc7	0x45	0xf4	0x00	0x00	0x00	0x00	0x83
0xec	0x04	0xff	0x35	0x60	0x99	0x04	0x08

- **Is this random?**

(NB: these are the 32 first bytes of `main()` in some x86 code)

- OK, even the human eye can see it
- Statistical entropy ≈ 1 bit apart: $\hat{H}_N^{MLE}(w) = 3.97641 \quad H_N(\mathcal{U}) = 4.87816$
- **This is not random:** std. dev ≈ 0.08 bit,

  → 99.9999% sure

Problem 2: confidence intervals

- Recognizing **text** as non-random: easy
- A bit more challenging:

0x55	0x89	0xe5	0x83	0xec	0x58	0x83	0xe4
0xf0	0xb8	0x00	0x00	0x00	0x00	0x29	0xc4
0xc7	0x45	0xf4	0x00	0x00	0x00	0x00	0x83
0xec	0x04	0xff	0x35	0x60	0x99	0x04	0x08

- **Is this random?**

(NB: these are the 32 first bytes of `main()` in some x86 code)

- OK, even the human eye can see it
- Statistical entropy ≈ 1 bit apart: $\hat{H}_N^{MLE}(w) = 3.97641 \quad H_N(\mathcal{U}) = 4.87816$
- **This is not random:** std. dev ≈ 0.08 bit, 99.9999% sure

Rather remarkable:
... we have only read **32** bytes
i.e., there are **224** values
we cannot have possibly seen
Extreme undersampling

Estimating standard deviation

Theorem [Antos, Kontoyiannis 2001]:
When N tends to +∞, $\sqrt{N}\ln 2(\hat{H}_N^{MLE} - H)$ is Gaussian with mean 0 and variance $\sigma_N^2 = Var\{-\log p(X)\}$

- Gives us no information for N small (yet)
- Non-degenerate case (variance ≠ 0) well-studied by statisticians
  ... but precisely,
  the uniform distribution **is** the degenerate case
- ... actually good news!

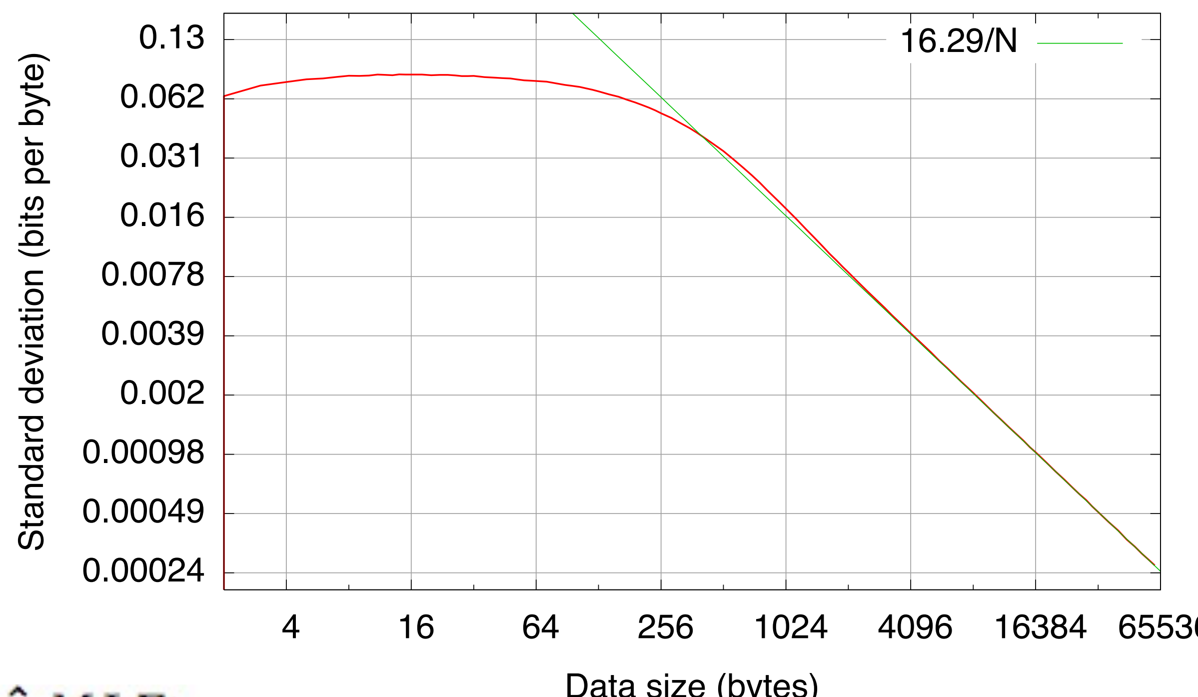
Estimating standard deviation

Theorem [Moddemeijer 2000]:

When N tends to $+\infty$, the std. dev. $SD(\hat{H}_N^{MLE}) \approx \sqrt{\sigma_N^2 + \frac{m-1}{2N^2 \ln^2 2}}$

(recall m=256)

- In the non-degenerate case, = O(1/$\sqrt{N}$)
- In the degenerate case, ≈ 16.29/N:
  much **smaller** (i.e., much better)
- N =32 bytes was about the worst case
  (std. dev ≈ 0.08)
- **99%** confidence interval is at 2.6 x $SD(\hat{H}_N^{MLE})$
  99.9% confidence interval is at 3.4 x $SD(\hat{H}_N^{MLE})$



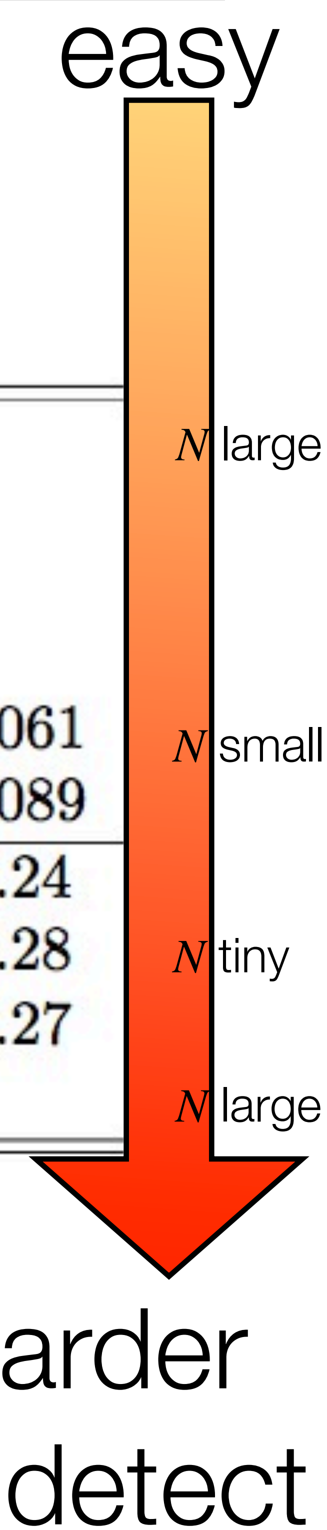
(Note: log-log scale)

Confidence intervals: practical experiments

- Experiments on **non-random** sources
- **99%** confidence intervals:
  (8.00 means 8±<0.01)
- **All entries** correctly classified

Data source	**Entropy** (bits/byte) $\hat{H}_N^{MLE}$	**Entropy** (bits/byte) H_N
Binary executable (elf-i386)	6.35	8.00
Shell scripts	5.54	8.00
Terminal activity	4.98	8.00
1 Gbyte e-mail	6.12	8.00
1Kb X.509 certificate (PEM)	5.81	7.80 ± 0.061
700b X.509 certificate (DER)	6.89	7.70 ± 0.089
130b bind shellcode	5.07	6.56 ± 0.24
38b standard shellcode	4.78	5.10 ± 0.28
73b polymorphic shellcode	5.69	5.92 ± 0.27
Random 1 byte NOPs (i386)	5.71	7.99

(code mutation)



Confidence intervals: practical experiments

- Experiments on **non-random** sources
- **99%** confidence intervals:
  (8.00 means 8±<0.01)
- **All entries** correctly classified

Data source	Entropy (bits/byte)	
	$\hat{H}_N^{MLE}$	H_N
Binary executable (elf-i386)	6.35	8.00
Shell scripts	5.54	8.00
Terminal activity	4.98	8.00
1 Gbyte e-mail	6.12	8.00
1Kb X.509 certificate (PEM)	5.81	7.80 ± 0.061
700b X.509 certificate (DER)	6.89	7.70 ± 0.089
130b bind shellcode	5.07	6.56 ± 0.24
38b standard shellcode	4.78	5.10 ± 0.28
73b polymorphic shellcode	5.69	5.92 ± 0.27
Random 1 byte NOPs (i386)	5.71	7.99

(code mutation)

Pretty remarkable:
shellcode is **encrypted,**
except tiny decryption routine
suffices to recognize it
as **non-random**

easy

N large

N small

N tiny

N large

harder
to detect

Outline

1. A few **scary stories** about computer security

2. **ORCHIDS**: an intrusion prevention system

3. **Semantics** and algorithms

4. **NetEntropy**: detecting subverted cryptographic flows

5. Conclusion

Outline

1. A few **scary stories** about computer security

2. **ORCHIDS**: an intrusion prevention system

3. **Semantics** and algorithms

4. **NetEntropy**: detecting subverted cryptographic flows

5. Conclusion

Conclusion

- Two examples of **mathematical rigor** in intrusion detection
  - **ORCHIDS**: **semantics** («what») dictates **algorithms** («how»)
  - **NetEntropy**: **precise** estimators + confidence intervals

Theorems

- Of course mathematics will not solve all your problems!

  But it will help you understand **why** something works,
  and under **which** conditions/for **what** values of the parameters,

- A mathematical model may be **idealized**...
  This is a good start! And certainly better than no model at all