Binary compatibility on NetBSD

Emmanuel Dreyfus, july 2014

About me

- Emmanuel Dreyfus <manu@netbsd.org>
- IT manager at ESPCI ParisTech as daylight job
- NetBSD contributor since 2001
- Milter-greylist since 2006
- OpenLDAP, glusterFS, mod_auth_mellon...
- *Le cahier de l'admin BSD*, Eyrolles editions

Binary compatibility

- Same CPU, different OS
- No emulation
- Kernel masquarade as target kernel
- Useful to run proprietary apps
- Almost native performances

What do we need?

- Identifying foreign binaries
- System calls translation
- Signals translation
- Nothing more...
- … except if non Unix-like target

Identifying aliens

- This happens inside execve(2)
- OS-specific dynamic linker?
  - .interp ELF section (see next slide)
  - objdump -s -j .interp /bin/ls
  - NetBSD: /libexec/ld.elf_so
  - Linux: /lib/ld-linux.so.2
- Only for dynamic binaries

Playing with objdump(1)

```
$  objdump -h /bin/ls
(...)
Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       00000013  0000004001c8  0000004001c8  000001c8  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.netbsd.ident 018  0000004001dc  0000004001dc  000001dc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.netbsd.pax 00014  0000004001f4  0000004001f4  000001f4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .hash         0000019c  000000400208  000000400208  00000208  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .dynsym       00000600  0000004003a8  0000004003a8  000003a8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
(…)

$ objdump -s -j .interp  /bin/ls
(...)
Contents of section .interp:
 4001c8 2f6c6962 65786563 2f6c642e 656c665f  /libexec/ld.elf_
 4001d8 736f00                               so.
```

Identifying *static* aliens

- OS-specific ELF section?
- List: objdump -h /bin/ls
- Dump, looking for OS-specifics
  - .comment
  - __libc_atexit
  - .gnu_debuglink
- This quickly turns into heuristics

System call tables

- Each process has a struct proc
- OS behavior described by struct emul
- Each struct emul has a system call table
- Each table defined in a syscalls.master fille:
  - src/sys/kern/syscalls.master
  - src/sys/compat/linux/arch/i386/syscalls.master
  - src/sys/compat/freebsd/syscalls.master
- Used to generate .c and .h files

Inside a syscall table

NetBSD native

```
0   INDIR          { int|sys||syscall(int code, \
                       ... register_t args[SYS_MAXSYSARGS]); }
1   STD            { void|sys||exit(int rval); }
2   STD            { int|sys||fork(void); }
3   STD    RUMP    { ssize_t|sys||read(int fd, void *buf, size_t nbyte); }
4   STD    RUMP    { ssize_t|sys||write(int fd, const void *buf, \
                     size_t nbyte); }
5   STD    RUMP    { int|sys||open(const char *path, \
                     int flags, ... mode_t mode); }
6   STD    RUMP    { int|sys||close(int fd); }
```

Linux i386

```
0   NOARGS       { int|linux_sys||nosys(void); } syscall
1   STD          { int|linux_sys||exit(int rval); }
2   NOARGS       { int|sys||fork(void); }
3   NOARGS       { int|sys||read(int fd, char *buf, u_int nbyte); }
4   NOARGS       { int|sys||write(int fd, char *buf, u_int nbyte); }
5   STD          { int|linux_sys||open(const char *path, int flags, \
                   int mode); }
6   NOARGS       { int|sys||close(int fd); }
```

System call translation

```
int
linux_sys_creat(struct lwp *l,
    const struct linux_sys_creat_args *uap,
    register_t *retval)
{
        /* {
                syscallarg(const char *) path;
                syscallarg(int) mode;
        } */
        struct sys_open_args oa;

        SCARG(&oa, path) = SCARG(uap, path);
        SCARG(&oa, flags) = O_CREAT | O_TRUNC | O_WRONLY;
        SCARG(&oa, mode) = SCARG(uap, mode);

        return sys_open(l, &oa, retval);
}
```

Difficult system call translation

- Features missing in native system
  - Sometimes the feature is just not exported
  - When NetBSD only had threads for Linux binaries
- Rewriting data buffer from userland
  - Once upon a time: stackgap security hazard
  - Nowadays: rewriting in kernel buffer
- Unbound data size
  - Causes multiple calls to underlying functions

Signals

- Catching a signal is a context switch
- Kernel must prepare a signal stack frame
- CPU-dependent, maybe with bits of assembly
- Trivial if identical to native version
- It may remain easy if close to native version
- Target behavior still needs to be analyzed

Easy CPU-specific signal code

src/sys/arch/powerpc/powerpc/linux_sigcode.S

```
#include <compat/linux/linux_syscall.h>

(...)
#define LINUX_SIGNAL_FRAMESIZE 64

#define SIGCODE_NAME      linux_sigcode
#define ESIGCODE_NAME     linux_esigcode
#define SIGNAL_FRAMESIZE LINUX_SIGNAL_FRAMESIZE
#define SIGRETURN_NAME   LINUX_SYS_sigreturn
#define EXIT_NAME         LINUX_SYS_exit

#include "sigcode.S"
```

Less easy CPU-specific signal code

src/sys/arch/i386/i386/freebsd_machdep.c

```
NENTRY(freebsd_sigcode)
   call    *FREEBSD_SIGF_HANDLER(%esp)
   leal    FREEBSD_SIGF_SC(%esp),%eax # scp (the call may have clobbered
                                      # the copy at SIGF_SCP(%esp))
   pushl   %eax
   pushl   %eax                       # junk to fake return address
   movl    $FREEBSD_SYS_sigreturn,%eax
   int     $0x80                      # enter kernel with args on stack
   movl    $FREEBSD_SYS_exit,%eax
   int     $0x80                      # exit if sigreturn fails
   .globl  _C_LABEL(freebsd_esigcode)
_C_LABEL(freebsd_esigcode):
```

Implementation how-to

1. Add entry in struct execsw
2. Add probe function to match foreign binaries
3. Create struct emul and system call table
4. Run, crash
5. Use ktrace(1), spot missing system call
6. Implement
7. Start over at step 4 until it works
8. At some time signals have to be implemented

Strange targets

- OS-specific system calls
- Non ELF based systems
  - PECOFF for Windows
  - Mach-O binaries on MacOS X
- Non Unix kernel interface
  - Win32 API for Windows
  - Mach microkernel on MacOS X

MacOS X binary compatibility

- Mach-O support
- Dual kernel (Mach+Darwin)
  - Two system call tables
  - Mach uses negative system calls
- Mach messages
- Mach ports, tasks, and rights

MacOS X oddities

- Mach microkernel
- /sbin/mach_init (later launchd) vs /sbin/init
- Mach microkernel
- IOKit and kernel servers
- Mach microkernel
- commpage

MacOS X compatibility successes

- MacOS X.3/PowerPC CLI tools working
- MacOS X.3/PowerPC Xdarwin fully functionnal
  - Client able to connect and operate
- WindowServer from MacOS X.2/PowerPC runs
  - But was replaced by QuartzDisplay in MacOS X.3

No happy end

- Never ran a binary on i386
- No Quartz client program ever displayed
- Little user interest
- No work beyond MacOS X.3
- Everything was cvs deleted

More informations

- Everything is at http://hcpnet.free.fr/pubz/
- OnLAMP papers on Linux and Irix compatibility
- EuroBSDcon 2004: MacOS X compatibility
- Last OS X compatibility status update: http://hcpnet.free.fr/applebsd

Questions?