Cleaning up the PCMCIA subsystem
lessons learned for kernel development

Dominik Brodowski
linux@dominikbrodowski.net

Instructor, Cybercrime and Criminal Procedure
Digital Forensics (Master studies)

Chair for European Criminal Law (Prof. Vogel), Tübingen
PCMCIA – long obsolete?

- state of the art for kernels 2.0, 2.2
- nowadays:
  - used by embedded systems
  - used on old systems
  - still present on moderately new hardware
old kernel-userspace interface

- kernel $\mapsto$ userspace: a card got inserted!
- userspace $\mapsto$ kernel: which card?
- kernel $\mapsto$ userspace: a networking card by manufacturer ABC.
- userspace $\mapsto$ kernel: try this driver!
- kernel $\mapsto$ userspace: OK, it worked!
Lesson 1: kernel-userspace interface

- backward-compatible API
- forward-compatible API
- simple, lean API
coding style

- typedef u_char cisdata_t;
- typedef struct dev_link_t { ...};
- #define CS_SUCCESS 0x00;
Lesson 2: Coding Style

- don’t hide information
- don’t duplicate information
- make the code readable to others
Lesson 3: trust code you control

- no in-kernel magic numbers
- no extra version numbers
driver probing, PCI style (simplified)

- The PCI core is informed about a new PCI device.
- Userspace is told about the new PCI device and its identification; userspace may load a module.
- It iterates over a doubly-linked list of all loaded drivers, and checks whether the driver’s ID table matches the IDs of the PCI device.
- It calls the “probe” callback of the PCI driver, which enables and activates the hardware.
Lesson 4: not invented here

- how do similar subsystems solve similar problems?
- share, but do not duplicate code
Lesson 5: all code runs on SMP

- locking, locking, locking
- reference counting and object lifetime
Lesson 6: „Keine Experimente!“

- proof of concept code: good that you have it
- experimental code: keep it separate, but show it to me
- code to be pushed: no experiments, please
Lesson 7: follow the kernel development model

- drivers/staging
- kernel-mentors
- development schedule?
**Lesson 8: you need the users**

- users do *some* of the testing for you
- but don’t make their lives to bad
Lesson 9: if things break, it’s always your fault

- fate of frontrunners
- unwanted side-effects
- unrelated changes
Lesson 10: *pick up some code and clean it up!*

- it is fun
- it is part of the Open Source approach
- it is rewarding
Thank You!

Feedback: linux@dominikbrodowski.net

Linux PCMCIA: http://www.kernel.org/pub/linux/utils
            ~~/kernel/pcmcia/pcmcia.html

Work: http://www.digitaleforensik.com/