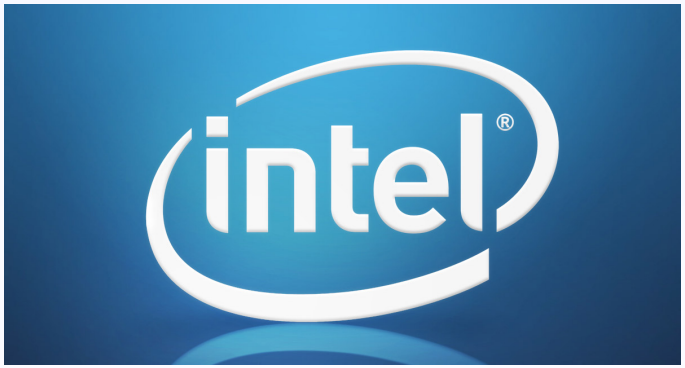
# Constructing Trusted Code Base XIII
## Methods of Trusted Execution

Aleksy Schubert & Jacek Chrząszcz



# Threats of malware: viruses, worms, troyans

- purpose: ransomware, spyware, adware, scareware, for fun, for experiment
- threats:
  - black-market exploitation
  - send email spam
  - contraband data hosting (e.g. child pornography)
  - DDoS nodes
  - extortion
  - monitoring of web browsing
  - display of unsolicited advertisements
  - redirect affiliate marketing revenues to the spyware creator
- typical path: WWW, email
- rotkits and backdoors implanted by malware

# Threats of malware: kinds of vulnerabilities

- security bugs:
  - buffer overruns (writes/reads/segfaults)
  - affecting control flow
  - incomplete control of access
- loopholes (aka features)
- design flaws (e.g. booting from external device, lack of sanitization)
- overpriviledged users (e.g. users with root access)
- over-privileged code (e.g. setuid programs, weak sandboxes)
- user error

# Preventing malware wrongdoing

- monitoring system actions (antivirus software, network alerts)
- scanning executables
- website security scans
- virtualization
- sandboxing
- jailing (system-level virtualization)
- no method for hardware implants

# Prevent from malware

- the value of software identity
- secure machines run only authorised code
- secure machines run only certified code
- ways to certify code
- ways to authorise code

# Break-once run anywhere

- all computers in network with the same system
- many machines in the world use the same system/software version
- diversity = higher maintenance cost = higher robustness

# Operating system vulnerabilities

- I/O processing: complex, full of exceptional control flows, tuning for performance can introduce threats
- vague access policy
- partial guarding of the access to resources due to efficiency reasons (incomplete mediation)
- hooks for extended security features

# Time of check, time of use

- in sharing time systems or distributed systems
- operation may be authorised at the beginning
- but its critical structures can be changed on the run
- runtime properties vs load-time properties
  - the policy may change

# Patching vulnerabilities

- can make the system less secure
  - narrow focus on the fault (immediate cause vs design flaw)
  - nonobvious side effects
  - failure somewhere else after fixing
  - non-fixing due to performance reasons

# Good design principles

- least privilege
- economy of mechanism (small, simple, and straightforward)
- open, public design (no security through obscurity)
- complete mediation
- conservative permission granting
- separation of privilege (many mechanisms to access)
- least common mechanism (weakest link principle)
- ease of use

# TCB as part of secure operating environment

- TCB can be ensured through:
  - hardware
  - some notion of processes (subjects)
  - primitive files (objects)
  - protected memory
  - some interprocess communication

# TCB to control security

- monitors four basic interactions:
  - process activation and switching
  - crossing execution domain boundaries
  - memory protection (secrecy and integrity for each domain)
  - guarding I/O operations

# Small TCB is possible

- Honeywell prototype secure operating system, *Scomp*
  - initially 20 routines
  - less than 1,000 lines of source code
  - final security kernel: 10,000 lines of code.
- seL4 microkernel: ca 10,000 lines of code