INFLUENTIAL OPERATING SYSTEM RESEARCH: SECURITY MECHANISMS AND HOW TO USE THEM

CARSTEN WEINHOLD
OVERVIEW

- Fundamental Concepts and Building Blocks
- Problems in Practice
- Security Architectures
- The Way Forward?
FUNDAMENTAL CONCEPTS
AND BUILDING BLOCKS
A capability is an unforgeable (immutable) token (piece of data) of authentication for some system resource possessed by a process. Possession itself grants access.

- First described by Dennis & Horn in 1966
- Managed and protected by system
- Attached to process, but cannot be forged by it
- Can be shared, transferred, inherited
An ACL is an out-of-process entity that allows to control access to some system resource. Access is granted after proactive checking by the enforcing system.

- Maps identifiers to access rights
- Attached to objects to be access by processes
- Managed by system, cannot be forged by process
- Usually whitelist, but can also include blacklist semantics
ROLE-BASED ACCESS CONTROL

- Programs always work in a role
- Program can drop to a lower role but not elevate to a higher role
- Higher role programs start lower role programs
- Roles can be selectively inheritable
- Example: SELinux
  - Every program effectively needs a policy
  - Huge maintenance burden and/or trust in vendor/distributor
- Combines all previous approaches (best of all worlds)
- Rules can be combined to sets
- Sets can be (selectively) inherited
PROCESSOR MODES

- Invented for MULTICS to enable multiprogramming environment
- Idea of rings of privilege that hold CPU instructions
- Inner rings can use instructions in the outer rings, but not vice versa
- Allows to implement memory protection via hardware-enforced addressing schemes
PROCESSOR MODES: X86-64

- Ring 3: User mode
- Ring 2: Unused
- Ring 1: Unused
- Ring 0: Kernel mode
- Ring "-1": Hypervisor mode
- Also:
  - System management mode (SMM)
  - Secure enclave ("SGX mode")
MEMORY PROTECTION

- Segmentation
- Paging
- Capability-Based Addressing
Flat (virtual) address space partitioning

First implemented in the Burroughs B5500, but also in MULTICS, IBM System/38, Intel 80286

Addresses relative to segment base register:
\[ \text{address} = \text{segment} + \text{offset} \]

Segment limit register marks size of segment

Memory segmentation visible to the process

RAM and file-system address spaces can be merged
- Hierarchical, per-process mapping of virtual memory to physical memory at page granularity
- First implemented in the Atlas Computer (1959/62), but also in IBM System/370, Intel IA-32 (since 80386), ...
- 2 (or 3+) protection domains: (VM) / Kernel / User-space
- Page sizes of limited variability (e.g., 4 kiB normal page and 4 MiB super page)
OS manages page tables
- Physical data layout and current consumption invisible to process
- Status and Permission bits in table entries specify access rights:
  - User vs kernel mode
  - Read/write/execute
CAPABILITY-BASED ADDRESSING

- Form of object-based addressing: every access to memory referenced through a capability
- No unrestricted pointer operations allowed in user space
- Single-address space possible ⇒ no context switches

Possible implementations:
- Store capabilities in protected memory area, modify through privileged process
- Extend memory with „capability bits“ to mark protected locations (recent example: CHERI capabilities)
PROBLEMS IN PRACTICE
- No (or only limited) protection within a process
- Programs can read / write within their own address space
- Use of pointers unsafe in: native code / C / C++, ...
Stack overflows smash may return addresses, jump anywhere on `ret`.

Overflows on heap may overwrite:
- Function pointers
- VTable pointers
- Memory management information

Partial mitigations:
- Canaries (but may be guessable)
- Shadow stacks for return addresses
- Write XOR execute semantics makes code injection attacks useless
- But return instruction still allows unrestricted jumps to arbitrary addresses: Return-oriented programming (ROP)
All security mechanisms are implemented / managed by the operating system kernel:
- Capabilities, ACLs, …
- Memory protection, …

All other OS functionality, too:

- Huge codebase, large attack surface exposed to applications
- All security mechanisms are implemented / managed by operating system kernel
  - Capabilities, ACLs, ...
  - Memory protection, ...

- All other OS functionality, too

- Huge codebase, large attack surface exposed to applications

- No protection within kernel
Not only malicious applications

Operating system kernel also exposed to untrusted input

- Network packets and protocols
- Thunderbolt, USB, other buses
- File-system images

One exploitable bug: kernel and all applications compromised
- Not only malicious applications
- Operating system kernel also exposed to untrusted input
  - Network packets and protocols
  - Thunderbolt, USB, other buses
  - File-system images
- One exploitable bug: kernel and all applications compromised
Microkernels split OS into small and isolated components

Better containment of faults and attacks (assuming safe interfaces)

But:

- Restricting interaction of components is still a big problem
- Does not help against physical attacks (access to device)
SECURITY ARCHITECTURES
Software integrity rooted in hardware:

- Only load and run software that matches a pre-determined checksum or public key (of the vendor)
- If software does not match, refuse to load
- Checksum or public key “fused” into hardware, cannot be exchanged

- Popular in system-on-chip (SOC) architectures, especially smartphones and tables

- Concept can be extended
- **Authenticated Booting:** Record the chain of trust in a tamper-proof hardware register, use as identity of loaded software stack.

- **Sealed memory:** Encrypt data such that it will only be released, if expected software is running.

- **Remote attestation:** Securely report identity to remote party.
- Challenger sends a random nonce to the system, he/she wants to have attested
- Challenged system responds with quote: identity of loaded software stack (PCR) + nonce, all signed using a private key
- Challenger check PCR signature based on known public key

Remote Attestation with Challenge/Response
HARDWARE-BASED ISOLATION

- Apple Security Processor
- ARM TrustZone
- Intel SGX
THE WAY FORWARD?
**Isolation Substrate:** Spatial and temporal isolation

**Legacy Codebase:** „old” code following monolithic design

**Trusted Component:** Smaller, more secure, or just „my own”

**Communication:** secure interaction between legacy codebase and trusted component
„Instead of vertically stacked libraries, we envision applications to be horizontal aggregates of communicating components, individually isolated from one another and mutually distrusting”

- Privileges of each component should be minimal (POLA)
EXAMPLE: SMART GRID

Smart Meter Appliance

- Legacy OS (Android)
- Smart Meter
- Attestation
- Microkernel
- TrustZone

Utility Server

- Attestation
- Anonymizer
- Legacy Database
- SGX

Attestation

TrustZone

Microkernel

SGX