InkTag: Secure Applications On An Untrusted Operating System

Owen S. Hofmann, Michael Z. Lee, Alan M. Dunn, Emmett Witchel
The University of Texas at Austin ***{osh,mzlee,adunn,witchel}@cs.utexas.edu***

Problem

OS vulnerablities are shared

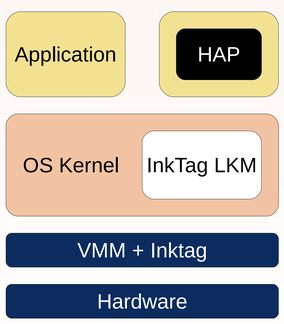
- Applications rely on large stack of system software
  - OS vulnerabilities become *shared* vulnerabilities
- Security-critical applications should have a way to isolate themselves and function without threat from the OS or other applications

Can applications run without OS trust?

- Most OS services have simple specification
  - Read last written data in files and address spaces
- A small hypervisor layer can provide privacy, verify integrity of OS-provided data
  - Previous systems: Overshadow, CHAOS, SP^3
  - Isolation, privacy, integrity for process address space and execution
- Many systems issues beyond isolation remain unaddressed
  - Naming (processes and files)
  - Access control policy
- Existing systems avoid OS interaction, must replicate OS data structures
  - Need memory map to authenticate page table updates

InkTag architecture

The InkTag VMM isolates *High-assurance processes* from the OS. The untrusted InkTag LKM tracks important process state, and communicates that state to the VMM.



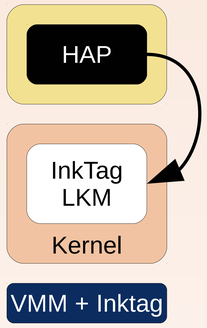
Memory mapping

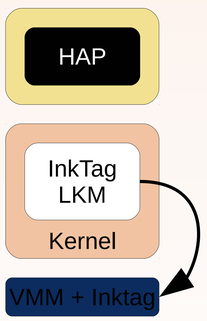
- InkTag must protect privacy and integrity for application address space
  - Only map those pages requested by application at the desired address
  - Protect order of pages in address space
- InkTag LKM registers Linux `pv_ops` interface to receive MMU updates
- InkTag VMM validates and installs address space updates

At mmap()

When creating new mapping, the HAP gives a *token* to the InkTag LKM to validate future page table updates. The InkTag LKM is untrusted, thus the token must be unforgeable.

$$HMAC(K_{HAP}, fileid, addr, end)$$

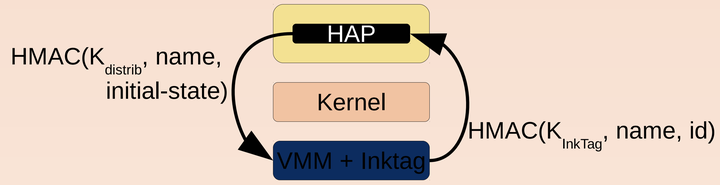




At page fault

When the kernel installs a new PTE, it calls the InkTag LKM via the `pv_ops` interface. The LKM passes the token to the InkTag VMM, which verifies that the new mapping is compatible with that requested by the HAP, and installs the PTE.

Identifying processes



- HAPs must be able to identify other applications
  - Cannot trust OS: PID or binary file name
- The distributer or administrator signs a hash of a HAP's initial state along with a *canonical name* (e.g. "/sbin/login")
- HAP passes signature to InkTag VMM at init
- VMM replies with signature of name and application instance id (a secure PID)

Access control

- OS information about access control state (user/group id) is untrusted
  - Do not want to import access control policy into hypervisor
- Delegate access control to HAPs: *Access control daemons* (ACDs)

When a HAP wishes to change principal (e.g. at login), it contacts an ACD. The ACD validates the request (possibly based on the canonical id) and passes the HAP a token for the HAP to prove to the VMM that the principal change is valid.