Advanced Systems Security: Securing Commercial Systems

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

February 18, 2010

Commercial Systems

- Focus on Performance, Flexibility, and Protection
- Do not satisfy the reference monitor concept
- But, lots of folks use them, so big potential benefit to making a commercial system secure
- Not so easy – so, lots of lessons learned and new ideas

Retrofitting Security

- Make legacy code satisfy the reference monitor concept
- Did the rules that we set depend on whether we built the reference validation mechanism from scratch or from legacy code?
- Which are hardest?
- Is it worth trying? Why?

Commercial Reference Monitors

- 1970s-80s: take an existing OS and add a reference validation mechanism
  - *KVM/370, VAX/VMS, Secure Xenix*
- 1980s-90s: Use microkernel architectures to deploy secure UNIX systems (like security kernel)
  - *DTMach, DTOS, Fluke/Flask*
- 1990s-2000s: UNIX systems for secrecy and integrity
  - *IX, DTE, LOMAC*

Early MLS Systems

- Data Secure UNIX and KSOS
- Compatible with the UNIX API
  - Emulation: UNIX calls implemented by KSOS kernel
  - Emulation with MLS had a significant performance impact
  - Scomp did not do emulation
- Insecure features of the UNIX interface presents problems too
  - Fork: when a new process is created, can share file descriptors
  - Why might this be a problem?

Early MLS Systems

- KVM/370
  - Virtual machine monitor for MLS systems
  - Retrofit of VM/370 for MLS
  - Indirect security-sensitive operations to the VMM
- Significant performance effect
  - 25% or more
  - Context switch to VMM
  - Reuse of VM/370 code limited optimizations

Early MLS Systems

- VAX/VMS
- DEC and Sandia Labs retrofit VAX/VMS for MLS
- Retrofit identified several vulnerabilities
- Prototype system

Early MLS Systems

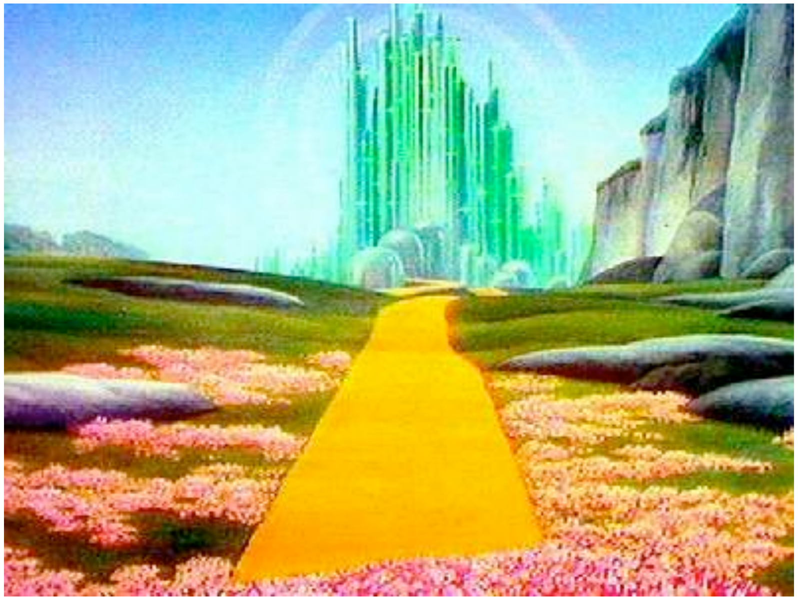
- Secure Xenix (later Trusted Xenix)
- Xenix: PC version of UNIX from Microsoft
- Goal: run Xenix apps without modification
- Two Problems
  - How do two instances of the same program running at different secrecy levels access the file system securely?
    - Consider /tmp: should a less secret version use a temp file created by the more secret version?
  - How does a user know that she is communicating with the trusted computing base?

Hidden Directories

- Now called *Polyinstantiated File Systems*
- Idea: There is a version of a directory for each secrecy level
- When a process opens a file, it opens the version based on its secrecy level
- How does this ensure MLS information flows?

Trusted Path

- Mechanism that enables a user to communicate with the system's trusted computing base
- E.g., Key attention sequence ctrl-alt-del
- Communication from user to TCB and from TCB to user
- Tough for windowing systems as we'll see



Microkernels and Security Kernels

- Microkernel systems emerged at this time, and they were gaining mindshare quickly as the way future OS's would be constructed
  - Mach microkernel
- Security kernel requirements:
  - Verifiable design
  - Map to implementation
- Should be easier for a microkernel than for a conventional kernel

Microkernel Architecture

PENNSTATE 1855

- Operating system consists of a *core kernel* component (microkernel) and a *set of servers* that implement traditional OS function
- Microkernels provide base function needed by all processing
    - Scheduling, IPC, Basic Device Access (IRQs), and others are discretionary
- OS Servers implement system specific function
    - Memory managers, file systems, networking, processes, naming, device drivers, advanced scheduling, …
- Idea: customize system on microkernel for function and performance – oh yeah, and security too

TMach, DTMach, DTOS

- MLS Mach systems
- Similar goals, but built by competing companies: *Trusted Information Systems* and *Secure Computing Corp*
- Approach: built MLS-aware servers on microkernel
- Applications would run at a single level
- Some also considered integrity
- A variety of innovations resulted (we'll take about some in DTE also)

Integrity

- The designers of these systems also considered protecting the integrity of computations
- For example, they envisioned a Clark-Wilson-like model where high integrity data would be modified by a sequence of high integrity operations
- How would they ensure that only these operations in that sequence would modify high integrity data?

Assured Pipelines

- *A sequence of high integrity processes that take input (high integrity) data to output*
- Use an MPS
- Input data is given a label
- Each process is given a unique label
- Each process's output is given a unique label
- Connect processes into a sequence based on data labels they input
- Use Type Enforcement for this

Mach Security Server

- Problem: Enable multiple, independent servers to enforce a coherent policy
  - How do server work together on security?
- Consider file opening
  - When a process requests opening of a file
  - Kernel authorizes process to access file server
  - The file server asks the security server if this access is authorized
  - The security server examines the policy and determines the answer

Kernel v. Server Enforcement

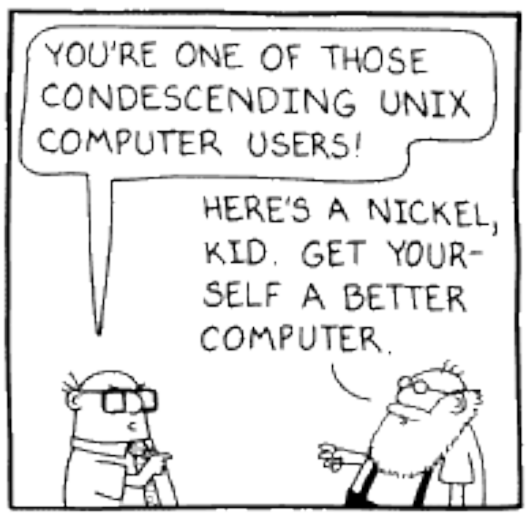
- Mach allows a party with the permission to send an IPC to a server to send any message to that server
  - Suppose a process can memory-map files into its address space via a memory server
  - Then, it can send any request to the memory server
  - Why is this a problem?
- Choice between enforcement layers and complexity
  - Kernel can enforce control if it understands the semantics of different messages
  - But, this requires more complexity in the kernel, but reduces the complexity (and trust required) in the server

What Happened to Microkernels?

- Second-generation microkernels emerged that made Mach obsolete
  - Fluke, L4, Exokernel, VINO, Eros, …
- But, no consensus (or market share for these microkernels), so these projects were not successful
  - Replaced by Linux
- Is the microkernel approach fundamental different?
  - How can we take advantage of that difference?
- Might this approach become popular?
  - Xen, L4, Minix 3, Proxos, … -- Windows is a microkernel

UNIX Systems

- Concurrent with microkernel exploration people continued the exploration of how to make a commercial OS MLS-secure?
  - These were focused on UNIX
  - Later, we'll talk about Trusted Solaris and Linux Security Modules, but first we talk about 2 systems…



IX

- AT&T Research UNIX prototype from the early 1990's – McIlroy and Reeds
- Goal: mandatory secrecy and integrity protection over file access
- Main focus is policy, although care was also taken in the construction of the trusted computing base

IX Model

- Dynamic information flow model with limits
- Protection state:
  - Each object and subject has a secrecy and an integrity label in each lattice that determines access (a la Denning)
- Labeling state
  - Traditional labeling based on creator
- Transition state
  - *High water mark* for secrecy limited by *ceiling*
  - *Low water mark* for integrity limited by *floor*
- Also, has support for assured pipelines as well

LOMAC Problem

- More mediation is required for dynamic policy enforcement than for static policy enforcement
- E.g., consider the case where a high integrity process accesses a high integrity file
  - That's OK
- But, then the process reads in a low integrity (malicious) executable file
  - We should lower the process's integrity, so now it cannot write to the high integrity file – hooray! Or oh no!
- But, what if the process mmapped the file?

Domain and Type Enforcement

- A UNIX prototype that controls setuid privilege escalation
- How does setuid normally work in UNIX?
- Why can that be a problem?

Domain and Type Enforcement

- Model – distinguish subject and object types
  - Subjects are domain types
  - They have access rights to objects and other domains (as before conceptually)
- And they are associated with entry point programs, which trigger the domain when run
  - This defines a transition state for the system – a refined setuid
- Consider passwd

Domain and Type Enforcement

- *Subject type*: user
- *Object type*: passwd_exec
- *Protection state:* user can execute passwd_exec
- *Transition state*: process exec'd from passwd_exec by user runs with passwd subject type

- This enables control of how a user runs a setuid program, and the permissions of the setuid process that results
- See this later in SELinux

Domain and Type Enforcement

- Another innovation that appeared here is
  - Labeled networking
- Consider a firewall
  - It tells you which ports can communicate with which IP addresses (and more)
  - But, they do not tell you which domains can access which ports – why is that important?
- IP Security Options (IPSO) adds fields for MLS labels in packet headers, which can be conveyed remotely
  - Is that enough?

Retrofit Evaluation

- **Complete Mediation**: Semi-formal op mediation?
  - This is certainly a goal
- **Complete Mediation**: Semi-formal for all classes?
  - Starting to see all classes with the addition of labeled networking to files in DTE
- **Complete Mediation**: Formal op mediation for all classes?
  - Not a particularly formal approach; difficult when retrofitting code

Retrofit Evaluation

- **Tamperproof**: Reference Monitor?
  - No methodological focus, although they took a look
- **Tamperproof**: TCB?
  - Harder yet
- **Verify**: Code?
  - This is not practical for retrofitted systems
  - Even Mach is too large
- **Verify**: Policy?
  - MLS and Integrity; TE enforces least privilege, but introduction of Assured Pipelines and LOMAC to UNIX systems improves matters

Take Away

- Hey, we have these systems with lots of marketshare – if only we had a secure version
- Motivated lots of efforts for retrofitting
- But, getting a system that does not satisfy the reference monitor concept to do that is not easy
  - Unsafe interfaces
  - Obtaining complete mediation
  - Verifiability is out the window – at this stage
- However, many concepts were discovered, such as Polymorphic FS, Assured Pipelines, Controlled Setuid