Language-Based Security

Greg Morrisett
Harvard University
Back in the early ‘90s, I was a graduate student at CMU. There were two very exciting groups:

- The PL group (SML)
- The OS group (Mach)

I hung out with people from both groups.
Microkernels

Operating systems in the 70’s were small and relatively trustworthy: “kernels”

Operating systems in the 80’s (e.g., Ultrix) were big, unreliable, inflexible.
– or glorified device drivers (e.g., DOS)

The Mach guys were going to fix all that…
My Mach Workstation

- DOS
- AFS Server
- Unix Server
- Paging Server
- X Server
- Emacs
- csh
- SML/NJ
- TCP/IP
- Micro-kernel
- disk
- network
- video

User space
The Key Idea:

Put everything possible at the user-level.

♦ Minimizes the Trusted Computing Base.
  – Less code has to be right to keep the system running and secure.
  – Isolates failures -- when DOS crashes, it shouldn’t affect my UNIX server.

♦ Flexible
  – Easy to add/remove new services.
  – Multiple personalities.
  – Application-specific services.
Flexibility: User-Level Paging

- SML/NJ wants to do a copying collection.
- After GC, the “old” space is no longer needed.
- A conventional pager would write those pages to disk and read them back in for the next GC.
- Our custom pager just dropped the pages and provided a zero-filled page at next GC.
Today’s Systems

- The OS is millions of lines of code.
  - Much of it is 3rd party (drivers, modules).
  - All written in C or assembly.
  - No isolation -- one buffer overrun ruins everyone’s day.

- Most of the user-level processes and DLLs run with the same privileges.
  - So there’s no real isolation.
  - I don’t know what half of them can do.
  - Many more things are “executable”
    - ps, pdf, doc, xls, html, …
What went wrong?

♦ There are other ways to be flexible
  – Dynamically loadable drivers & modules.
  – Hypervisor (e.g., vmware) can give us multiple personalities.

♦ Performance
  – Twice as many context switches.
  – Twice as much copying.
  – It’s very easy to measure performance.

♦ Who cares?
  – Not so easy to measure reliability/security.
  – Back then, attacks weren’t frequent because the stakes economic weren’t that high.
SysCall in Monolithic Kernel

**User Process**
calls `f=fopen("foo")`
library executes "break"
calls `fread(f,n,&buf)`
library executes "break"

**Kernel**
saves context, flushes TLB, etc.
checks UID against ACL, sets up IO buffers & file context, pushes ptr to context on user’s stack, etc.
restores context, clears supervisor bit
saves context, flushes TLB, etc.
checks f is a valid file context, does disk access into local buffer, copies results into user’s buffer, etc.
restores context, clears supervisor bit

Time
SysCall in Mach

User Process

f=fopen("foo")
“break”

Kernel

saves context
checks capabilities, copies arguments
switches to Unix server context

Unix Server

checks ACL, sets up buffers, etc.
“returns” to user.

saves context
checks capabilities, copies results
restores user’s context
Nothing special about OSs

- Strong isolation is always a good idea from a security & reliability standpoint.
- But the costs always seem to lead us to tear down the walls:
  - Originally, web servers forked a separate process with attenuated capabilities for each CGI request.
  - The overheads of the fork were seen as too expensive, so the script is run in the context of the server…
  - Same story for web clients, databases, …
The Challenges

We need a new architecture for security.

♦ We need better security policies.
  – Centralized, end-to-end: my credit card should not go out the door without my consent, and never in the clear and never to an untrusted 3rd party.
  – The policies need to be phrased at the level of applications and humans, not OS objects or keys.
  – They need to be simple and easily [re]configured.

♦ We need better enforcement mechanisms.
  – Avoid the overheads that tempt us back into a monolithic mind set.
  – Minimize the amount of code or data that we have to trust.
This Course

Software-based policy enforcement:

♦ Dynamic Isolation (SFI)
♦ Static Isolation (PCC)
♦ Certifying Compilation (TAL)
♦ Legacy code (Stackguard, Ccured, Cyclone)
♦ Authorization (Stack Inspection, IRMs)
♦ Confidentiality (SLam, Jif)
Language-Based Security

The topics I’m covering are really a subset of language-based security (software security.)

There are many other exciting areas where languages, analysis, compilers, and semantics are informing security:

♦ Policy languages and logics (e.g., BAN logic)
♦ Models and protocols (e.g., Spi)