Mobile Code

- Shipping computation from one host to another is a very useful paradigm
  - Applets: programs can be more compact than equivalent data, can interact with user with low latency
    - Can be used for complex GUIs, page description languages, etc.
  - Agents: program acting on behalf of a user, can interact with its environment with low latency
    - Can be used for data collection (e.g. price comparison), load-balancing, long-lived computing tasks
  - Servlets, ASPs: code submitted by clients that would like to run in the context of a larger software system
    - Web servers, rent-a-server, database systems, etc.

Mobile Code
Protection within a Single Address Space

Emin Gun Sirer

Problems

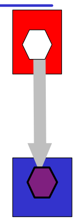
- Mobile code is invaluable in building extensible systems
- But in general, running code provided by someone else poses a security risk
- Could place every extension in a separate hardware address space
  - The code could perform any read, write, jump operation and the MMU would catch any missteps
  - The OS could catch every system call and direct through a reference monitor
  - BUT, the extension code typically must run in the same protection domain as the base system to be useful

Mobile Code Protection

- Can we place extension code in the same address space as the base system, yet remain secure ?
  - Imagine how an app can modify the paging policy the OS uses for its pages
- Many techniques have been proposed
  - SFI
  - Safe interpreters
  - Language-based protection
  - PCC

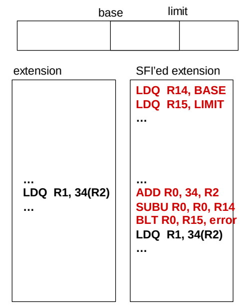
SFI

- Control what the application can do by managing the instruction stream
- Software fault isolation (SFI)
  - Assign a range of contiguous addresses to each extension
  - Rewrite the extension's code segment, inserting checks before every read, write and jump to ensure that it is legitimate
- Checks can be cheap
  - Need only recompute address and perform range check, 3-7 instructions



SFI Loads and Stores

- Every load and store is preceded by the check that the hardware would have done
- Dedicate two general purpose registers to hold the base and limit
- Modern processors have extra stall cycles during which the checks can be performed



SFI control flow

- An extension should only be able to jump to well-defined entry points in the system
- Restrict control flow to indirect jumps off of a table

SFI

- Hard to share data
  - Must still be copied from one extension's memory range into another's
- Performance problems
  - The checks extract a high penalty
- Hard to scale to large numbers of extensions

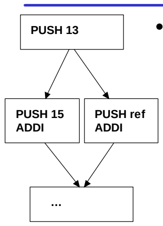
Safe Interpreters

- Restrict code to an interpreted language
  - E.g. telescript, python, perl, tcl, etc…
- The application must go through interpreter for execution
  - The interpreter can enforce security checks at any instruction, the application does not have direct access to hardware
- Slow

Language-based Typesafety

- Constrain the vocabulary of the extensions to a subset of safe instruction sequences
  - Force the programmer to use a language that cannot express unsafe operations
- Many instances
  - Imperative: Java, Modula-3, Limbo
  - Functional: ML, O'caml, Haskell
  - Domain-specific: BPF
- Use a *verifier* to check statically that extensions will not violate safety at runtime

Verification



- Verifier is a specialized theorem-prover
  - System safety depends on axioms such as "thou shalt not create arbitrary pointers through pointer arithmetic"
  - Verifier simulates all possible executions of the program, making conservative assumptions
  - Checks for violation of safety axioms

PCC

- Proof-carrying code
- Extension peresents a certificate that it is safe w.r.t. a safety policy
  - Certificate is a proof in first-order logic
  - The proof is linked to the code
  - The recipient evaluates the proof to check if the safety condition holds over the program
- Details beyond scope of this OS course
  - See courses by Prof. Morisett and Prof. Kozen