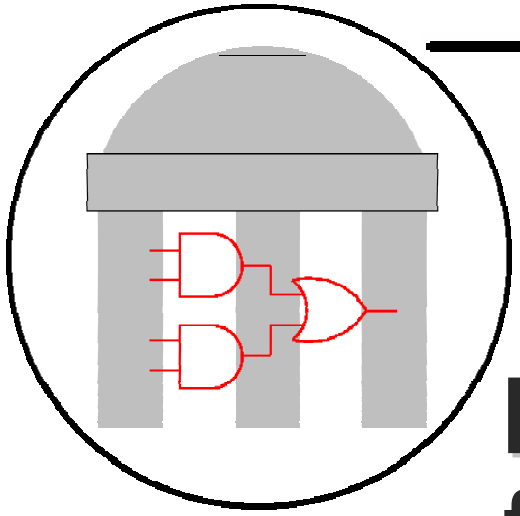


Program Partitioning for Secure Execution

Charles W. O'Donnell
G. Edward Suh
Srini Devadas

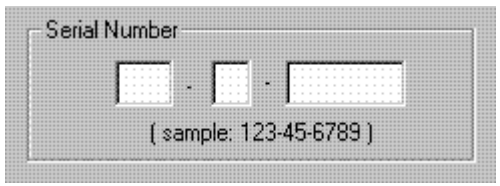
September 24, 2004

4th MIT CSAIL Computer Architecture Workshop

Licensing

- $software Licensing important
  - Software making money ⇒ Jobs after graduation
  - Alternatives problematic
    - Service instead of software
    - Free software, bug support

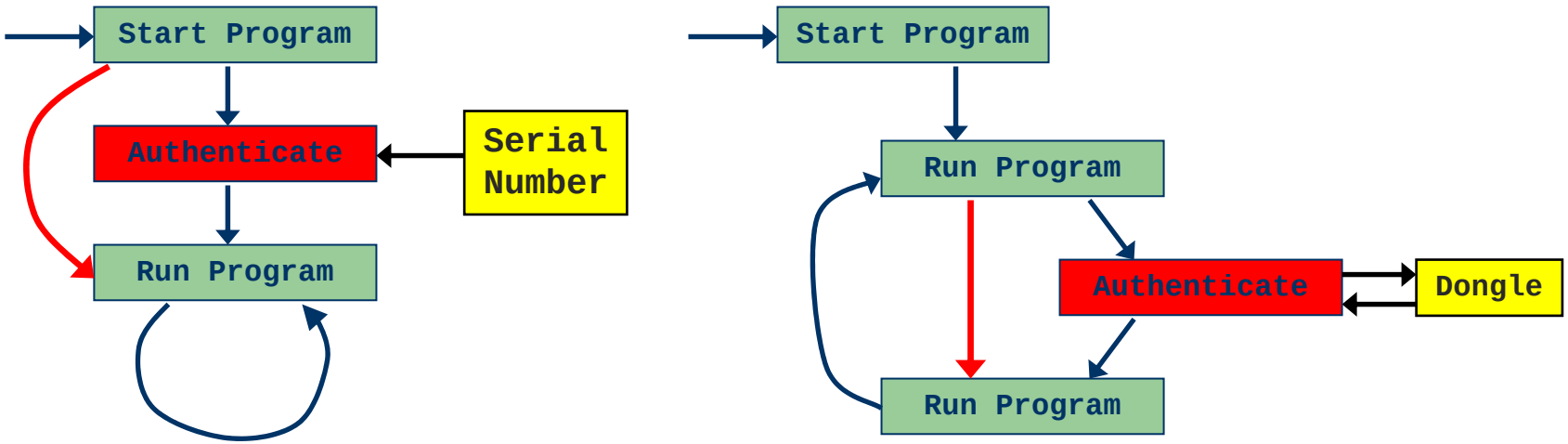
- Past methods poor
  - Online License Authentication
  - Serial Numbers
  - Dongles



Polling No Good

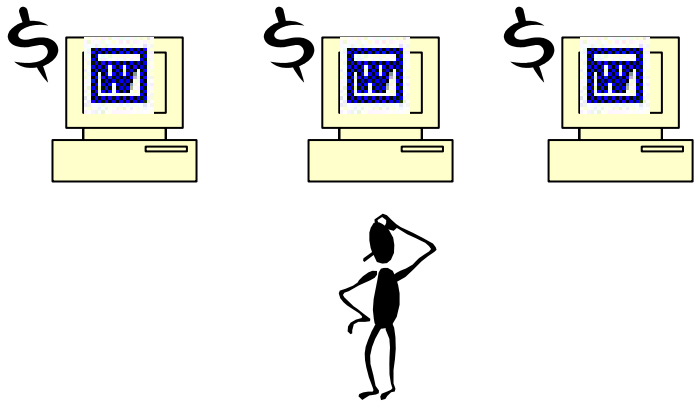
- Dongle / Online Verification simply **polling**
  - "Checks" are not critical to application's functionality
  - Easily bypassed

Control Flows:



Secure CPUs?

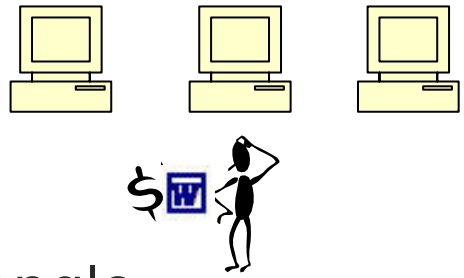
- **AEGIS** to the rescue?
  - Encrypt application, tie to CPU/machine
  - People use many machines
- Tie License to a Person?

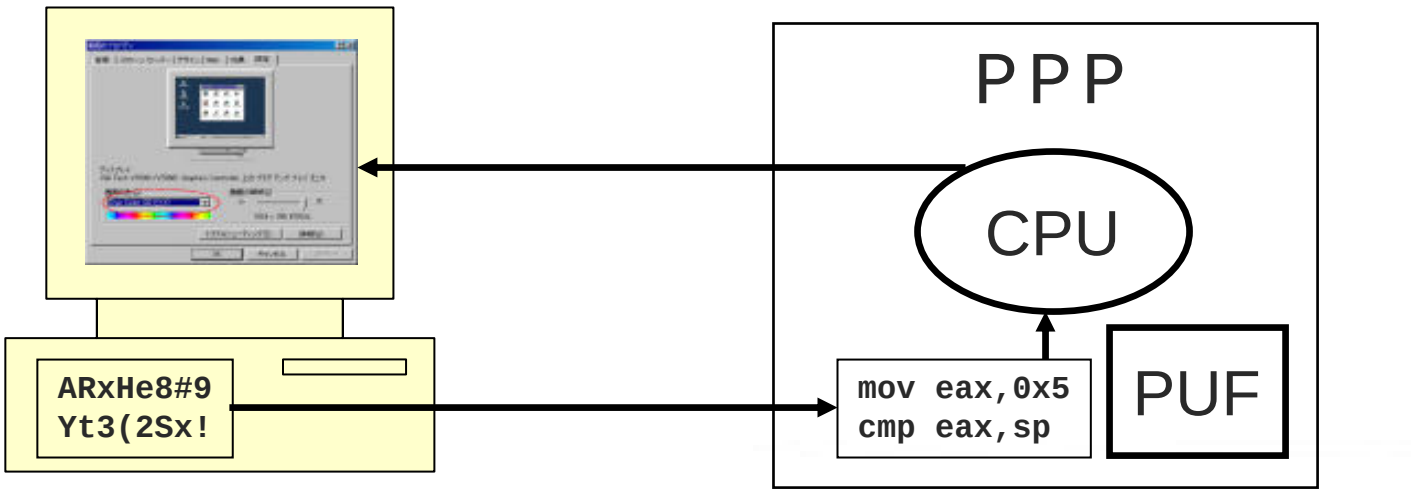


- Major problems with Secure CPUs?
  - They're slow
  - They don't run very fast
  - (Seriously, simply can't beat "normal" CPUs)

Portable Protected Processor (PPP)

- Secure CPU (entire trust base) on a single dongle
  - Solely identified as yours (PUF)
  - Put on your keychain
  - Dumb terminals execute all code via PPP (wholly encrypted just for you) using this dongle





But this would be even ***slower***

Only use PPP when necessary

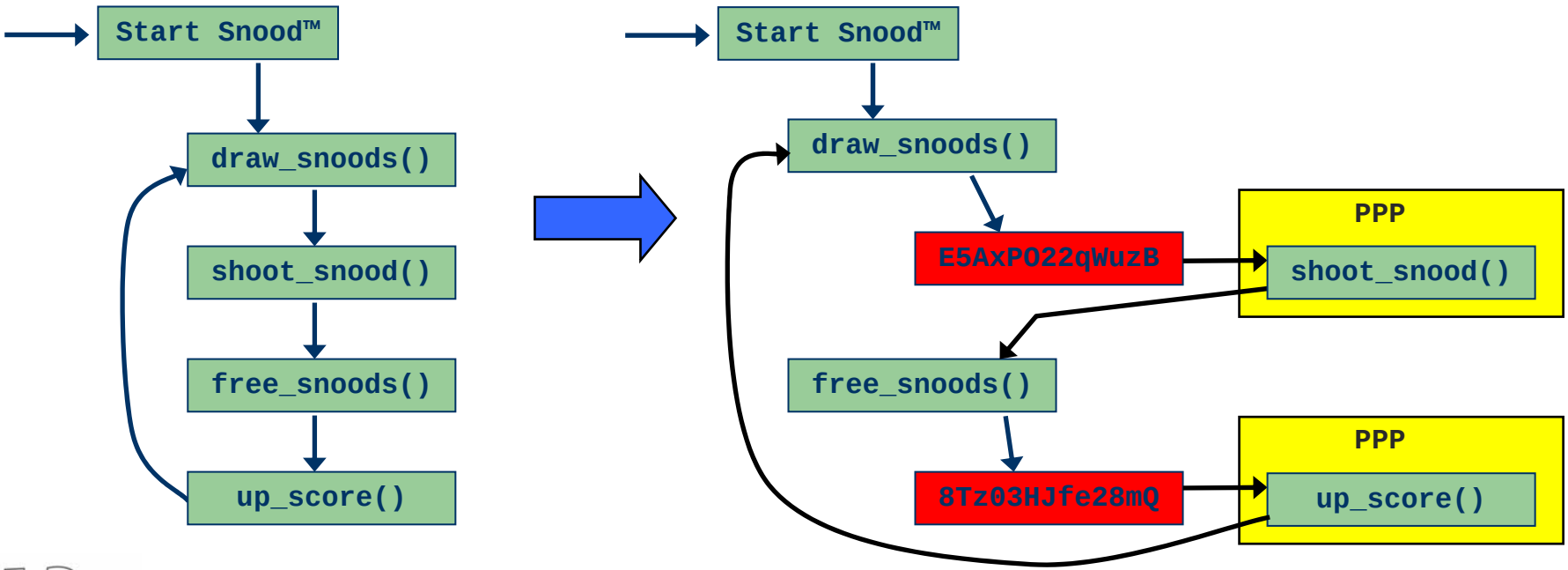
- "Check" PPP like old dongles
  (But way cooler because smarter?)
  - Fundamentally same problem

- Minimal requirements:
  (1) Protected Program ***requires*** your PPP to work
  (2) Protected Program ***can encrypt*** some proprietary algorithms

Program Partitioning

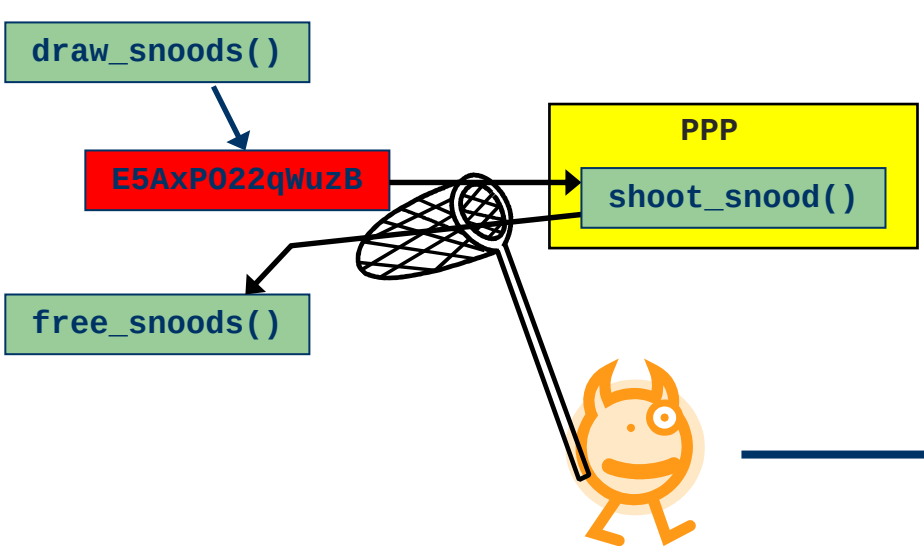
- Partition the program into ***plain*** and ***encrypted*** text

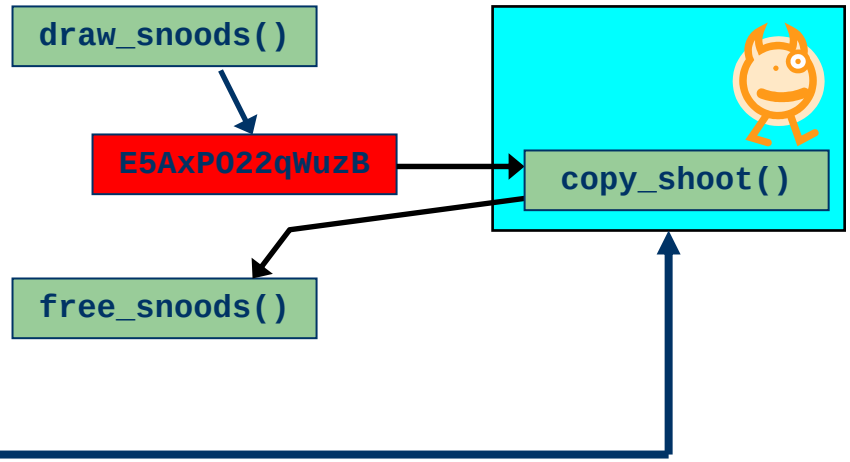
  Execute encrypted text (only) on PPP
  - Encrypted "code" is not useless (like dongle checks) but critical to the execution of the application
  - Can also encrypt proprietary algorithms



Partitioning Balance

- Encrypted regions on critical path 
  Encrypted regions small, imitate-able 

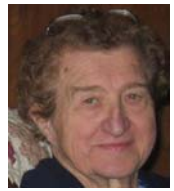




- ***Balance*** what portions to encrypt, which not to
  - Less encryption ⇒ faster
  - More encryption ⇒ harder to reverse-engineer

Solution Requirements

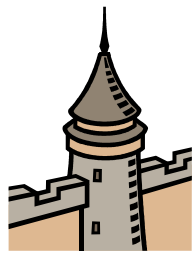
- *Some* solution should answer…
  - Adversarial Model



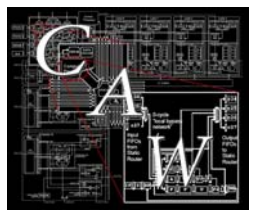
or

  - *Parameterizable* methodology for different adversaries
  - Metric for “level of security”

  - Transition point strategy and attack repulsion

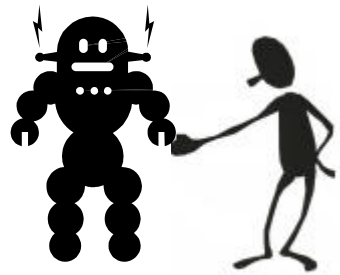


  - Architectural modifications for speed



Transitions: Toy ideas

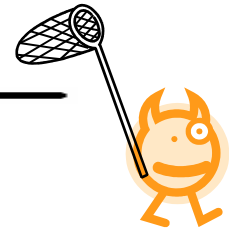
- Compiler & Human Determined
  - Easier with compiler in trust-base
  - Language-level definition for proprietary algorithms



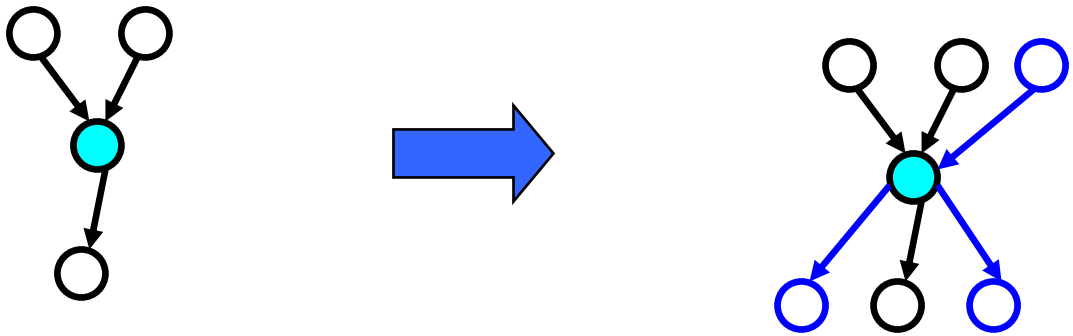
- Control flow graph interpretation
  - Compiler determination of critical paths

- Monitor code addition
  - Interwoven with required code

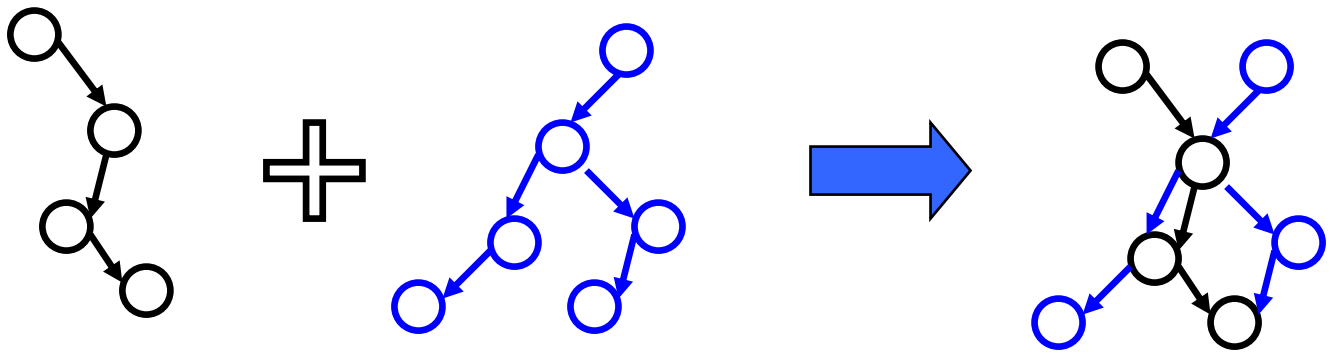
Attack Complexity: Toy ideas



- Obfuscation techniques
  - Increases Ingress and Egress count



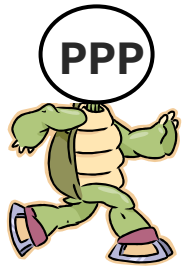
- Control flow graph unionizing with unrelated flow
  - Similar to watermarking techniques



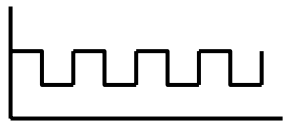
- Data flow graph manipulations

Architectural Modifications: Toy Ideas

- PPP has slow interface
  - Try to keep/reuse data within PPP
  - But simplifies attack



- Need very fast switching between Host PC and PPP



- Might need extra context or tagged awareness
  - Host CPU multitasking

Beyond Licensing

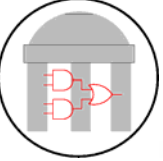
- PPP uniquely identifies ***you*** with application
  - Personalized settings
  - Secure Identity situations (online purchase, etc)



- Need stronger ***privacy*** model
  - End-to-end protection of data leaving PPP

Summary

- PPP tied to license of software
- Partition encrypted and plain text of application software along critical path
- Compiler & Human determined transformations to defend attacks and protect IP
- Architectural changes needed for efficiency
- Extending PPP gives more possibilities



Thanks