Introduction to Information Security

0368-3065, Spring 2015

Lecture 5:
Confinement (2/2)

Eran Tromer

Slides credit:
Dan Boneh and John Mitchell, Stanford

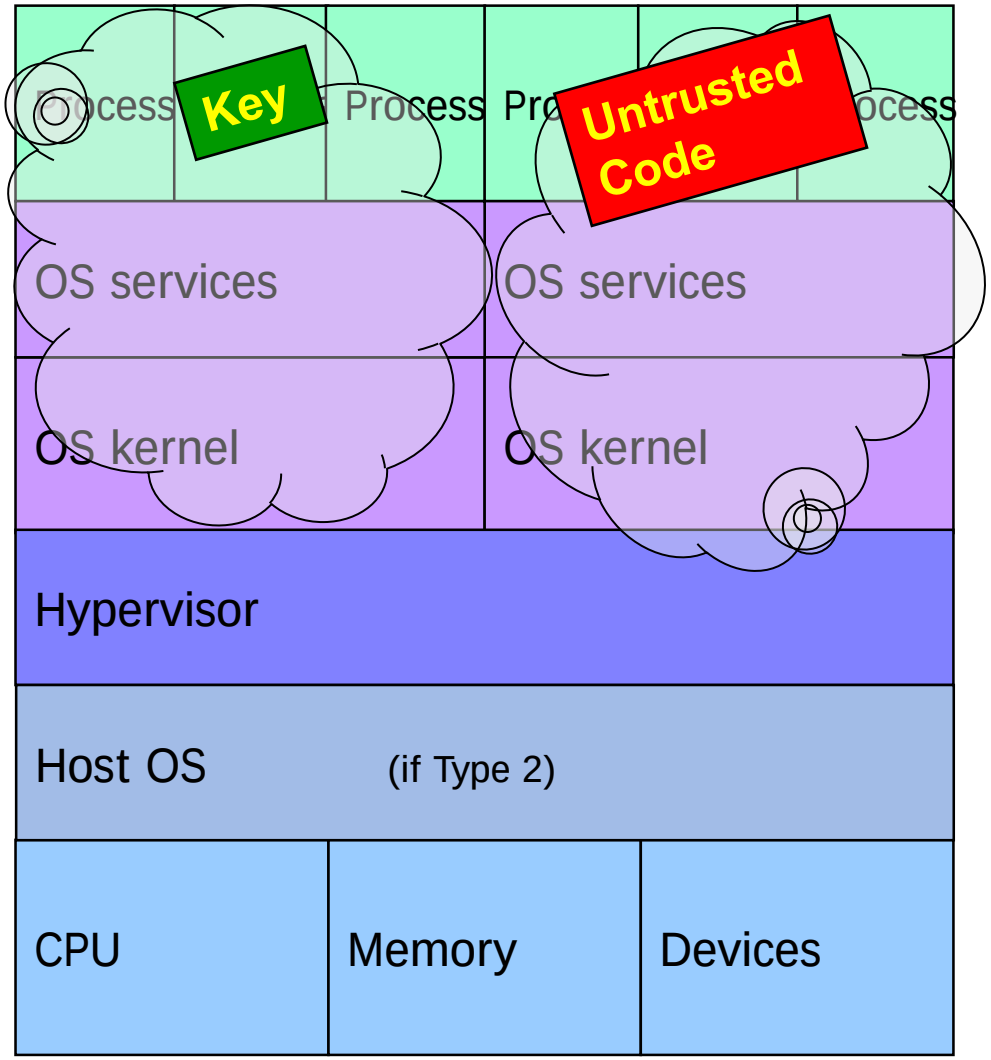
Confinement using Virtual Machines

Virtual machines

Security benefits:

- **Confinement** (isolation, sandboxing)
- Management
- Monitoring
- Recovery
- Forensics (replay)

US patent 6,922,774 (NSA NetTop)



VMM security assumption

- VMM Security assumption:
  - Malware may infect guest OS and guest apps
  - But malware cannot escape from the infected VM
    - Cannot infect host OS
    - Cannot infect other VMs on the same hardware
- Requires that VMM protect itself and is not buggy
  - VMM is much simpler than full OS
  - VMM API is much simpler than OS API
  - (but host OS still has device drivers)

Example of VM security application:

VMM Introspection

protecting the anti-virus system

Example: intrusion Detection / anti-virus

- Runs as part of OS kernel and user space process
  - Kernel root kit can shutdown protection system
  - Common practice for modern malware
- Standard solution: **run IDS system in the network**
  - Problem: insufficient visibility into user's machine
- Better: **run IDS as part of VMM (protected from malware)**
  - VMM can monitor virtual hardware for anomalies
  - VMI: Virtual Machine Introspection
    - Allows VMM to check Guest OS internals

Sample checks

Stealth malware:

- Creates processes that are invisible to "ps"
- Opens sockets that are invisible to "netstat"

1. Lie detector check
   - Goal: detect stealth malware that hides processes and network activity
   - Method:
     - VMM lists processes running in GuestOS
     - VMM requests GuestOS to list processes (e.g. ps)
     - If mismatch, kill VM

Sample checks (cont.)

2. Application code integrity detector
   - VMM computes hash of user app-code running in VM
   - Compare to whitelist of hashes
     - Kills VM if unknown program appears

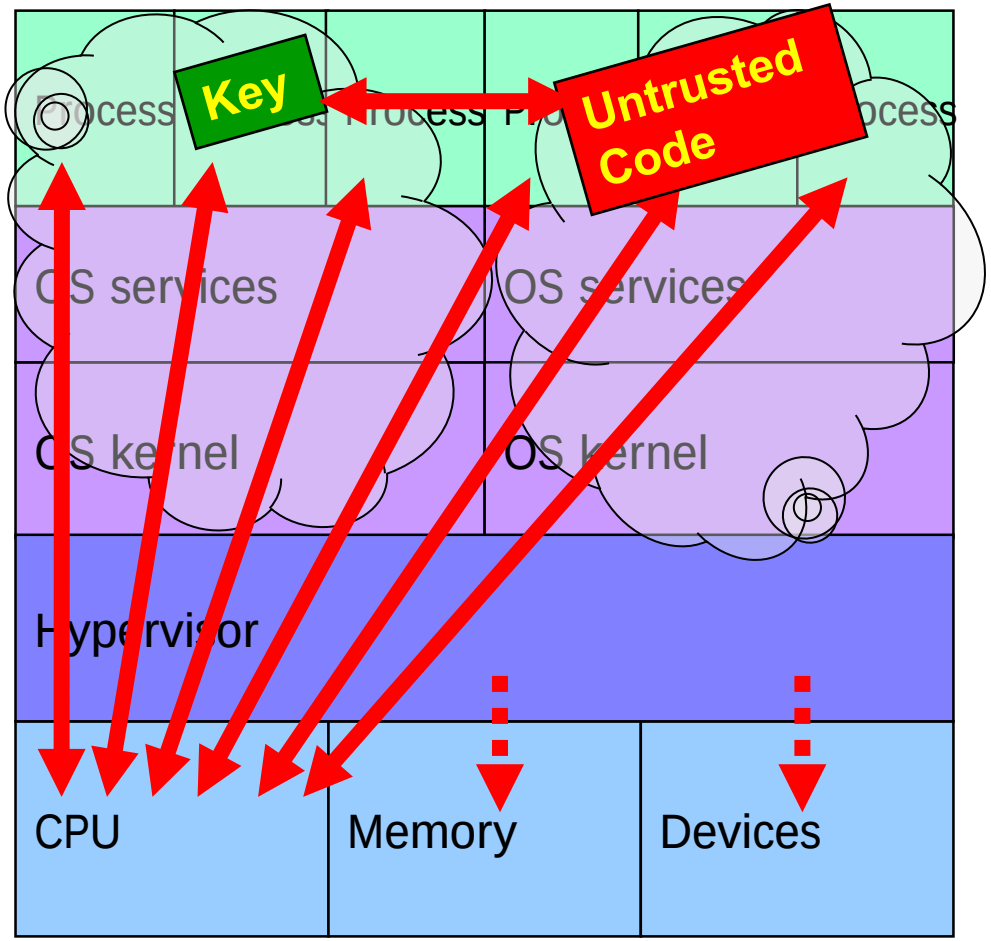
3. Ensure GuestOS kernel integrity
   - example: detect changes to sys_call_table

4. Virus signature detector
   - Run virus signature detector on GuestOS memory

5. Detect if GuestOS puts NIC in promiscuous mode

Covert channels and **side channels** in virtualization

- Covert channel: unintended communication channel between isolated but *cooperating* components (*sender* and *receiver*)
  - Can be used to leak classified data from secure component to public component
- Side channel: unintended channel that lets an *attacker* component retrieve information from an *victim* component without the latter's cooperation
- Often induced by low-level resource contention

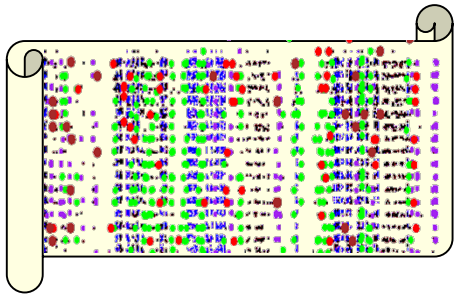


An example covert channel

- Both VMs use the same underlying hardware
- To send a bit $b \in \{0,1\}$ malware does:
  - b= 1: at midnight do CPU intensive calculation
  - b= 0: at midnight do nothing
- At midnight, listener does a CPU intensive calculation and measures completion time
  - Now b = 1 $\Leftrightarrow$ completion-time > threshold
- Many covert channel exist in running system:
  - File lock status, cache contents, interrupts, …
  - Very difficult to eliminate

Cache-based side-channels in cloud computing

Demonstrated, using Amazon EC2 as a study case:



- Cloud cartography
  Mapping the structure of the “cloud” and locating a target on the map.

- Placement vulnerabilities
  An attacker can place his VM on the same physical machine as a target VM (40% success for a few dollars)

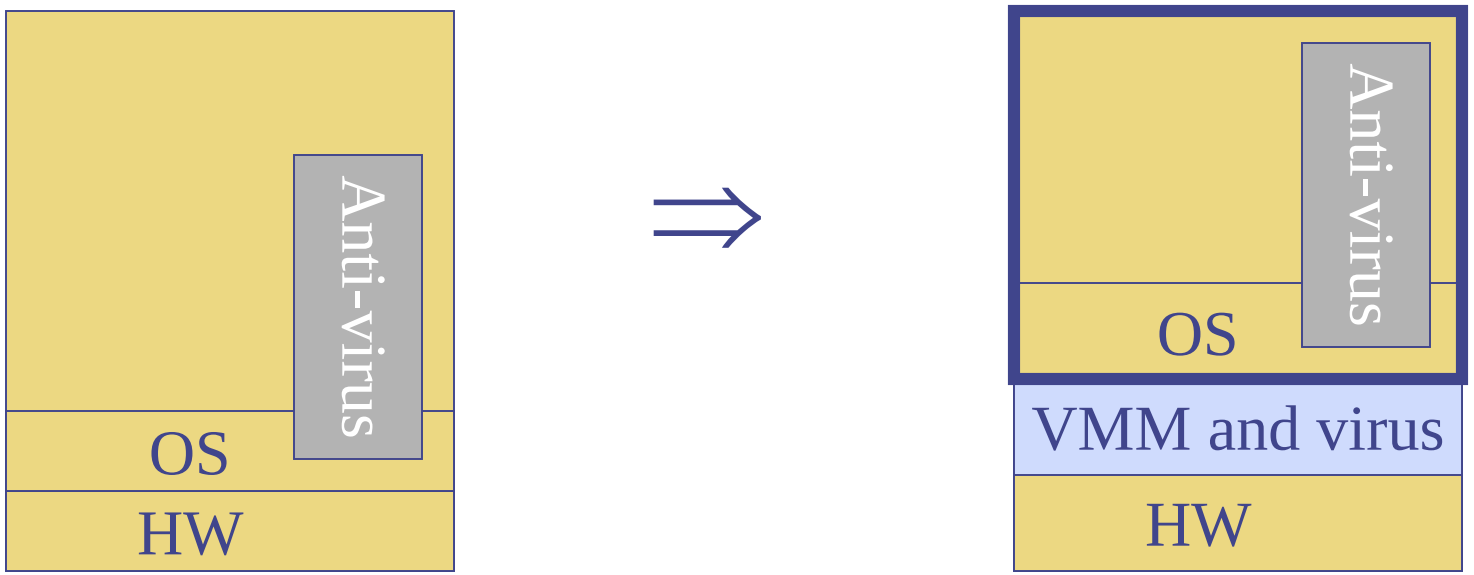
- Cross-VM exfiltration
  Once VMs are co-resident, secret information can be exfiltrated across VM boundary.
  (Simulated: theft of decryption keys!)

Motivation

Virtual machine confinement:
a blessing or a curse?

Subvirt [King Chen Wang Verbowski Wang Lortch 2006]

- Virus idea:
  - Once on the victim machine, install a malicious VMM
  - Virus hides in VMM
  - Invisible to virus detector running inside VM



The MATRIX


The Red Pill
The Blue Pill

VM Based Malware (blue pill virus)

[Rutkowska 2006]

- A virus that installs a malicious VMM (hypervisor) on-the-fly under running OS
- Use SVM/VT-x to create VM
- Microsoft Security Bulletin: (Oct, 2006) :
  Suggests disabling hardware virtualization features by default for client-side systems
  http://www.microsoft.com/whdc/system/platform/virtual/CPUVirtExt.mspx
- VMBRs are easy to defeat
  - A guest OS can detect that it is running on top of VMM

VMM Detection

- Can an OS detect it is running on top of a VMM?
- Applications:
  - Virus detector can detect VMBR
  - Normal virus (non-VMBR) can detect VMM
    - refuse to run to avoid reverse engineering
  - Software that binds to hardware (e.g. MS Windows) can refuse to run on top of VMM
  - DRM systems may refuse to run on top of VMM

VMM detection (red pill techniques)

1. VM platforms often emulate simple hardware
   - VMWare emulates an ancient i440bx chipset

     ... but report 8GB RAM, dual Opteron CPUs, etc.
2. VMM introduces time latency variances
   - Memory cache behavior differs in presence of VMM
   - Results in relative latency in time variations for any two operations
3. VMM shares the TLB with GuestOS
   - GuestOS can detect reduced TLB size: repeatedly access *k* pages and detect for what *k* thrashing occurs.
4. Deduplication (VMM saves single copies of identical pages)

... and many more methods **[GAWF'07]**

VMM Detection

Bottom line: **The perfect VMM does not exist**

- VMMs today (e.g. VMWare) focus on:

  Compatibility: ensure off the shelf software works

  Performance: minimize virtualization overhead

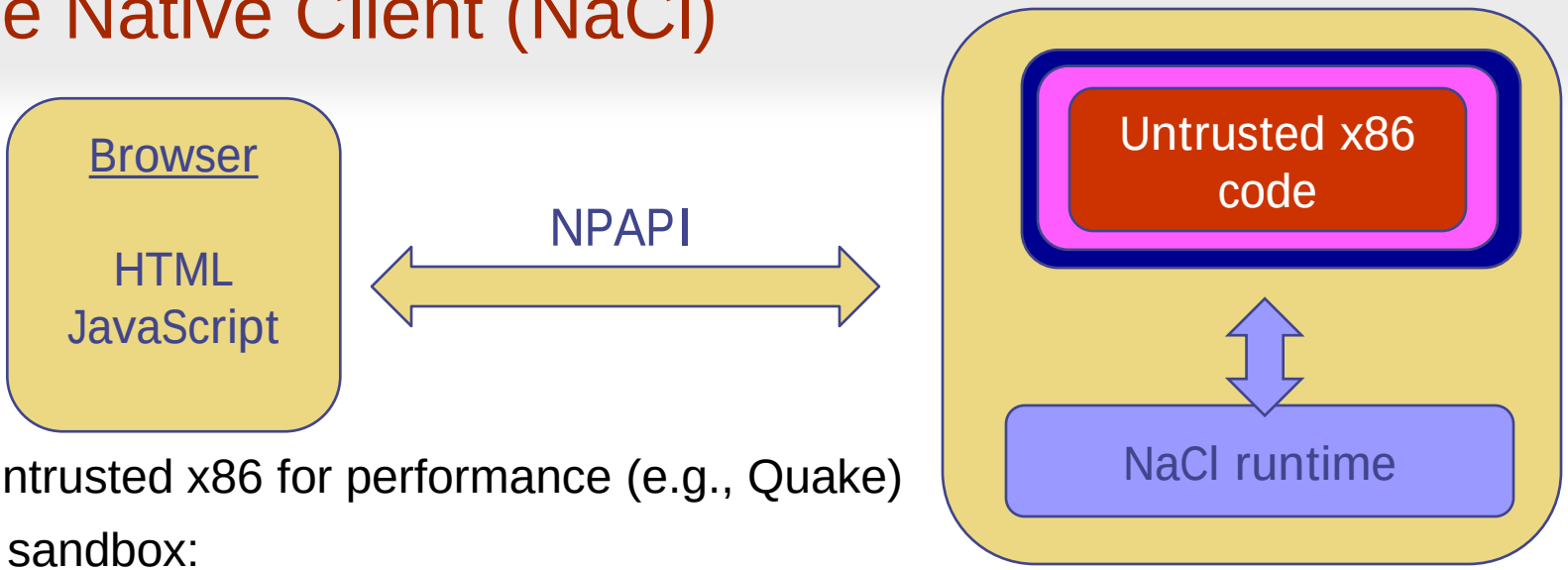
- VMMs do not provide **transparency**
  - **Anomalies reveal existence of VMM**

Application-level confinement

Application-level confinement

- Interpreters for non-native code
  - JavaScript, Java Virtual Machine, .NET CLR, Flash
- Isolating threads sharing same address space:
  - Software Fault Isolation (SFI), e.g., Google Native Code

Userspace sandboxing using Software Fault Isolation: Google Native Client (NaCl)



- Run untrusted x86 for performance (e.g., Quake)
- Outer sandbox:
  - Process confinement
  - Restricts capabilities using system call interposition (on Linux: uses `seccomp` to tell the kernel to filter its system calls)
- Inner sandbox:
  - Uses x86 memory segmentation to isolate application memory from one another
  - Restricts allowed machine code to protect the segmentation
    - Allows only a subset of x86
    - All jumps are to aligned blocks (to prevent jumping into middle of op)
  - All inter-process communication and services are through NaCl runtime
  - Verified during loading

Process confinement: Linux security modules

Linux Security Modules (in-kernel)

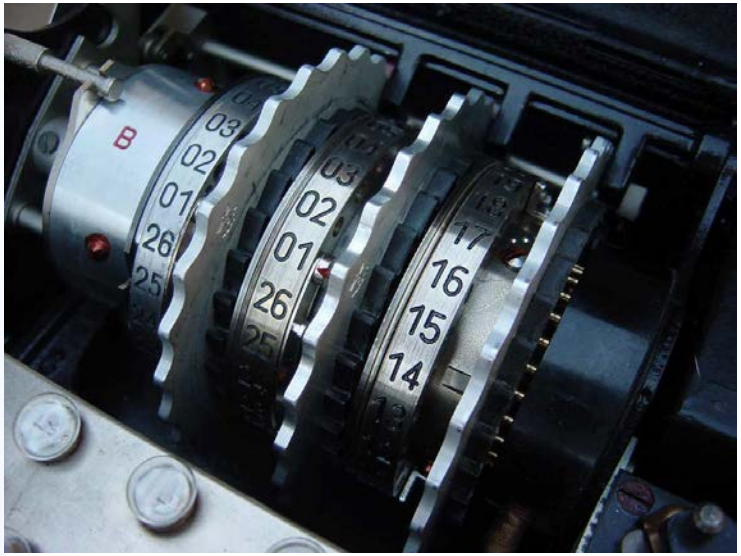
- SELinux
  - Role-based user/process labels
  - Object labels
    - Identifies filesystem objects by inode and associated extended attribute (set via system policy)
  - Policy determines who can access what
  - Default policies, can be configured
    - Change policy and relabel filesystem
    - Manually relabel file system objects
- AppArmor
  - Identified filesystem objects by path
- Smack
  - Goal: simplicity

All are Mandatory Access Control.

Cryptography overview

History of crypto

- Ceaser cipher
- Electromechanical ciphers (e.g., Enigma)
- Information theory
- Complexity theory
- Modern cryptography



Cryptography

- Pros
  - A tremendous tool
  - The basis for many security mechanisms
  - Provides formally-defined security-guarantees under formally-specified assumptions
- Cons
  - Not the solution to all security problems
    - Functionality
    - Assumptions on adversary's power
    - Performance
  - Relies on unproven (but wildly believed and extensively analyzed) computational assumtions
  - Unreliable unless implemented properly
  - Unreliable unless used properly
- Do not try to invent/implement yourself unless
  - you spend a lot of time becoming an expert
  - you subject your design to outside review

Scenarios

- Storage
  - Store files privately
  - Protect files from tampering
- Communication
  - Avoid eavesdropping
  - Avoid corruption
  - “Secure channel”
- Authentication
- Many protocols