CS162
Operating Systems and Systems Programming
Lecture 27

Protection and Security II, ManyCore Operating Systems

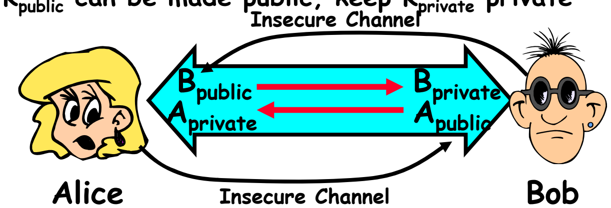
December 8, 2008

Prof. John Kubiatowicz

http://inst.eecs.berkeley.edu/~cs162

Review: Public Key Encryption Details

- Idea: K_{public} can be made public, keep $K_{private}$ private



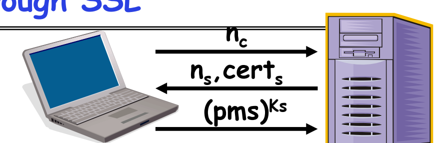
- Gives message privacy (restricted receiver):
  - Public keys can be acquired by anyone/used by anyone
  - Only person with private key can decrypt message
- What about authentication?
  - Alice→Bob: [(I'm Alice)^{Aprivate} Rest of message]^{Bpublic}
  - Provides restricted sender and receiver
- Suppose we want X to sign message M?
  - Use private key to encrypt the digest, i.e. $H(M)^{Xprivate}$
  - Send both M and its signature:
    - » Signed message = [M,$H(M)^{Xprivate}$]
  - Now, anyone can verify that M was signed by X
    - » Simply decrypt the digest with X_{public}
    - » Verify that result matches H(M)

Goals for Today

- Use of Cryptographic Mechanisms
- Authorization Mechanisms
- Worms and Viruses

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Also, slides on Taint Tracking adapted from Nickolai Zeldovich

Security through SSL



- SSL Web Protocol
  - Port 443: secure http
  - Use public-key encryption for key-distribution
- Server has a certificate signed by certificate authority
  - Contains server info (organization, IP address, etc)
  - Also contains server's public key and expiration date
- Establishment of Shared, 48-byte "master secret"
  - Client sends 28-byte random value n_c to server
  - Server returns its own 28-byte random value n_s, plus its certificate $cert_s$
  - Client verifies certificate by checking with public key of certificate authority compiled into browser
    - » Also check expiration date
  - Client picks 46-byte "premaster" secret (pms), encrypts it with public key of server, and sends to server
  - Now, both server and client have n_c, n_s, and pms
    - » Each can compute 48-byte master secret using one-way and collision-resistant function on three values
    - » Random "nonces" n_c and n_s make sure master secret fresh

Recall: Authorization: Who Can Do What?

- How do we decide who is authorized to do actions in the system?
- **Access Control Matrix:** contains all permissions in the system
  - Resources across top
    - » Files, Devices, etc…
  - Domains in columns
    - » A domain might be a user or a group of permissions
    - » E.g. above: User D_3 can read F_2 or execute F_3
  - In practice, table would be huge and sparse!

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- Two approaches to implementation
  - Access Control Lists: store permissions with each object
    - » Still might be lots of users!
    - » UNIX limits each file to: r,w,x for owner, group, world
    - » More recent systems allow definition of groups of users and permissions for each group
  - Capability List: each process tracks objects has permission to touch
    - » Popular in the past, idea out of favor today
    - » Consider page table: Each process has list of pages it has access to, not each page has list of processes …

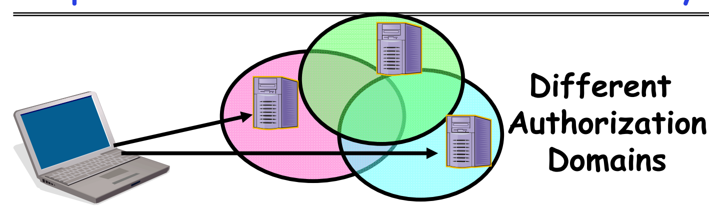
How fine-grained should access control be?

- Example of the problem:
  - Suppose you buy a copy of a new game from "Joe's Game World" and then run it.
  - It's running with your userid
    - » It removes all the files you own, including the project due the next day…
- How can you prevent this?
  - Have to run the program under *some* userid.
    - » Could create a second *games* userid for the user, which has no write privileges.
    - » Like the "nobody" userid in UNIX – can't do much
  - But what if the game needs to write out a file recording scores?
    - » Would need to give write privileges to one particular file (or directory) to your *games* userid.
  - But what about non-game programs you want to use, such as Quicken?
    - » Now you need to create your own private *quicken* userid, if you want to make sure tha the copy of Quicken you bought can't corrupt non-quicken-related files
  - **But – how to get this right??? Pretty complex…**

Authorization Continued

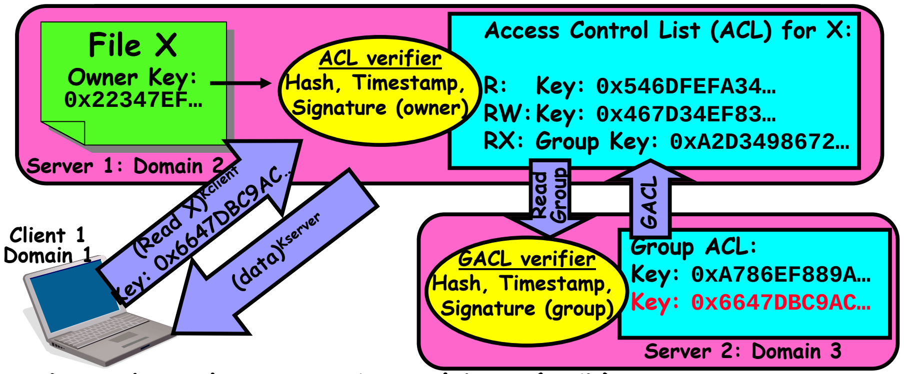
- **Principle of least privilege:** programs, users, and systems should get only enough privileges to perform their tasks
  - Very hard to do in practice
    - » How do you figure out what the minimum set of privileges is needed to run your programs?
  - People often run at higher privilege then necessary
    - » Such as the "administrator" privilege under windows
- One solution: Signed Software
  - Only use software from sources that you trust, thereby dealing with the problem by means of authentication
  - Fine for big, established firms such as Microsoft, since they can make their signing keys well known and people trust them
    - » Actually, not always fine: recently, one of Microsoft's signing keys was compromised, leading to malicious software that looked valid
  - What about new startups?
    - » Who "validates" them?
    - » How easy is it to fool them?

How to perform Authorization for Distributed Systems?



- Issues: Are all user names in world unique?
  - No! They only have small number of characters
    - » kubi@mit.edu → kubitron@lcs.mit.edu → kubitron@cs.berkeley.edu
    - » However, someone thought their friend was kubi@mit.edu and I got very private email intended for someone else…
  - Need something better, more unique to identify person
- Suppose want to connect with any server at any time?
  - Need an account on every machine! (possibly with different user name for each account)
  - **OR: Need to use something more universal as identity**
    - » **Public Keys! (Called "Principles")**
    - » **People *are* their public keys**

Distributed Access Control



- Distributed Access Control List (ACL)
  - Contains list of attributes (Read, Write, Execute, etc) with attached identities (Here, we show public keys)
    - » *ACLs signed by owner of file, only changeable by owner*
    - » *Group lists signed by group key*
  - ACLs can be on different servers than data
    - » *Signatures allow us to validate them*
    - » *ACLs could even be stored separately from verifiers*

Analysis of Previous Scheme

- Positive Points:
  - Identities checked via signatures and public keys
    - » *Client can't generate request for data unless they have private key to go with their public identity*
    - » *Server won't use ACLs not properly signed by owner of file*
  - No problems with multiple domains, since identities designed to be cross-domain (public keys domain neutral)
- Revocation:
  - What if someone steals your private key?
    - » *Need to walk through all ACLs with your key and change…!*
    - » *This is very expensive*
  - Better to have unique string identifying you that people place into ACLs
    - » *Then, ask Certificate Authority to give you a certificate matching unique string to your current public key*
    - » *Client Request: (request + unique ID)$^{C_{private}}$; give server certificate if they ask for it.*
    - » *Key compromise⇒must distribute "certificate revocation", since can't wait for previous certificate to expire.*
  - What if you remove someone from ACL of a given file?
    - » *If server caches old ACL, then person retains access!*
    - » *Here, cache inconsistency leads to security violations!*

Analysis Continued

- Who signs the data?
  - Or: How does the client know they are getting valid data?
  - Signed by server?
    - » What if server compromised? Should client trust server?
  - Signed by owner of file?
    - » Better, but now only owner can update file!
    - » Pretty inconvenient!
  - Signed by group of servers that accepted latest update?
    - » If must have signatures from all servers ⇒ Safe, but one bad server can prevent update from happening
    - » Instead: ask for a threshold number of signatures
    - » Byzantine agreement can help here
- How do you know that data is up-to-date?
  - Valid signature only means data is valid older version
  - Freshness attack:
    - » Malicious server returns old data instead of recent data
    - » Problem with both ACLs and data
    - » E.g.: you just got a raise, but enemy breaks into a server and prevents payroll from seeing latest version of update
  - Hard problem
    - » Needs to be fixed by invalidating old copies or having a trusted group of servers (Byzantine Agrement?)

Administrivia

- Midterm II: Still grading
  - *Solutions are up*
  - Will be back by Wednesday (I hope)
  - Final date for regrade requests: Friday (12/12)
- Final Exam
  - December 18th,8:00-11:00AM, Bechtel Auditorium
  - Covers whole course (except last lecture)
  - Two pages of handwritten notes, both sides
- Last Day of Class – Next Wednesday
- Final Topics suggestions (so far). Obviously too many…
  - Quantum Computers (and factoring)
  - Mobile Operating Systems
  - Multicore Systems
  - Dragons
  - User Sessions
  - Power Management
  - Data Privacy
  - Berkeley OS History

Involuntary Installation

- What about software loaded without your consent?
  - Macros attached to documents (such as Microsoft Word)
  - Active X controls (programs on web sites with potential access to whole machine)
  - Spyware included with normal products
- Active X controls can have access to the local machine
  - Install software/Launch programs
- Sony Spyware [Sony XCP] (October 2005)
  - About 50 CDs from Sony automatically installed software when you played them on Windows machines
    - » Called XCP (Extended Copy Protection)
    - » Modify operating system to prevent more than 3 copies and to prevent peer-to-peer sharing
  - Side Effects:
    - » Reporting of private information to Sony
    - » Hiding of generic file names of form $sys_xxx; easy for other virus writers to exploit
    - » Hard to remove (crashes machine if not done carefully)
  - Vendors of virus protection software declare it spyware
    - » Computer Associates, Symantec, even Microsoft

Enforcement

- Enforcer checks passwords, ACLs, etc
  - Makes sure the only authorized actions take place
  - Bugs in enforcer⇒things for malicious users to exploit
- In UNIX, superuser can do anything
  - Because of coarse-grained access control, lots of stuff has to run as superuser in order to work
  - If there is a bug in any one of these programs, you lose!
- Paradox
  - Bullet-proof enforcer
    - » Only known way is to make enforcer as small as possible
    - » Easier to make correct, but simple-minded protection model
  - Fancy protection
    - » Tries to adhere to principle of least privilege
    - » Really hard to get right
- Same argument for Java or C++: What do you make private vs public?
  - Hard to make sure that code is usable but only necessary modules are public
  - Pick something in middle? Get bugs and weak protection!

State of the World

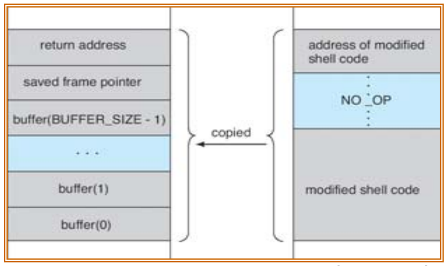
- State of the World in Security
  - Authentication: Encryption
    - » But almost no one encrypts or has public key identity
  - Authorization: Access Control
    - » But many systems only provide very coarse-grained access
    - » In UNIX, need to turn off protection to enable sharing
  - Enforcement: Kernel mode
    - » Hard to write a million line program without bugs
    - » Any bug is a potential security loophole!
- Some types of security problems
  - Abuse of privilege
    - » If the superuser is evil, we're all in trouble/can't do anything
    - » What if sysop in charge of instructional resources went crazy and deleted everybody's files (and backups)???
  - Imposter: Pretend to be someone else
    - » Example: in unix, can set up an .rhosts file to allow logins from one machine to another without retyping password
    - » Allows "rsh" command to do an operation on a remote node
    - » Result: send rsh request, pretending to be from trusted user→install .rhosts file granting you access

Other Security Problems

- Virus:
  - A piece of code that attaches itself to a program or file so it can spread from one computer to another, leaving infections as it travels
  - Most attached to executable files, so don't get activated until the file is actually executed
  - Once caught, can hide in boot tracks, other files, OS
- Worm:
  - Similar to a virus, but capable of traveling on its own
  - Takes advantage of file or information transport features
  - Because it can replicate itself, your computer might send out hundreds or thousands of copies of itself
- Trojan Horse:
  - Named after huge wooden horse in Greek mythology given as gift to enemy; contained army inside
  - At first glance appears to be useful software but does damage once installed or run on your computer

Security Problems: Buffer-overflow Condition

```
#define BUFFER SIZE 256
int process(int argc,
            char *argv[])
{
  char buffer[BUFFER SIZE];
  if (argc < 2)
       return -1;
  else {
       strcpy(buffer,argv[1]);
       return 0;
  }
}
```

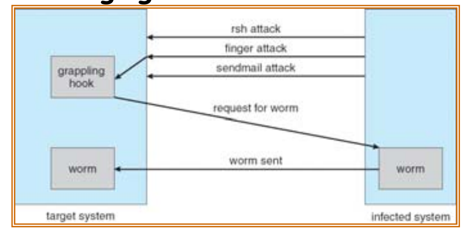


Before attack   After attack

- Technique exploited by many network attacks
  - Anytime input comes from network request and is not checked for size
  - Allows execution of code with same privileges as running program – but happens without any action from user!
- How to prevent?
  - Don't code this way! (ok, wishful thinking)
  - New mode bits in Intel, Amd, and Sun processors
    - » Put in page table; says "don't execute code in this page"

The Morris Internet Worm

- Internet worm (Self-reproducing)
  - Author Robert Morris, a first-year Cornell grad student
  - Launched close of Workday on November 2, 1988
  - Within a few hours of release, it consumed resources to the point of bringing down infected machines



- Techniques
  - Exploited UNIX networking features (remote access)
  - Bugs in *finger* (buffer overflow) and *sendmail* programs (debug mode allowed remote login)
  - Dictionary lookup-based password cracking
  - Grappling hook program uploaded main worm program

Some other Attacks

- Trojan Horse Example: Fake Login
  - Construct a program that looks like normal login program
  - Gives "login:" and "password:" prompts
    - » You type information, it sends password to someone, then either logs you in or says "Permission Denied" and exits
  - In Windows, the "ctrl-alt-delete" sequence is supposed to be really hard to change, so you "know" that you are getting official login program
- Salami attack: Slicing things a little at a time
  - Steal or corrupt something a little bit at a time
  - E.g.: What happens to partial pennies from bank interest?
    - » Bank keeps them! Hacker re-programmed system so that partial pennies would go into his account.
    - » Doesn't seem like much, but if you are large bank can be millions of dollars
- Eavesdropping attack
  - Tap into network and see everything typed
  - Catch passwords, etc
  - Lesson: never use unencrypted communication!

Timing Attacks: Tenex Password Checking

- Tenex – early 70's, BBN
  - Most popular system at universities before UNIX
  - Thought to be very secure, gave "red team" all the source code and documentation (want code to be publicly available, as in UNIX)
  - In 48 hours, they figured out how to get every password in the system
- Here's the code for the password check:

```
for (i = 0; i < 8; i++)
  if (userPasswd[i] != realPasswd[i])
    go to error
```

- How many combinations of passwords?
  - 256^8?
  - Wrong!

Defeating Password Checking

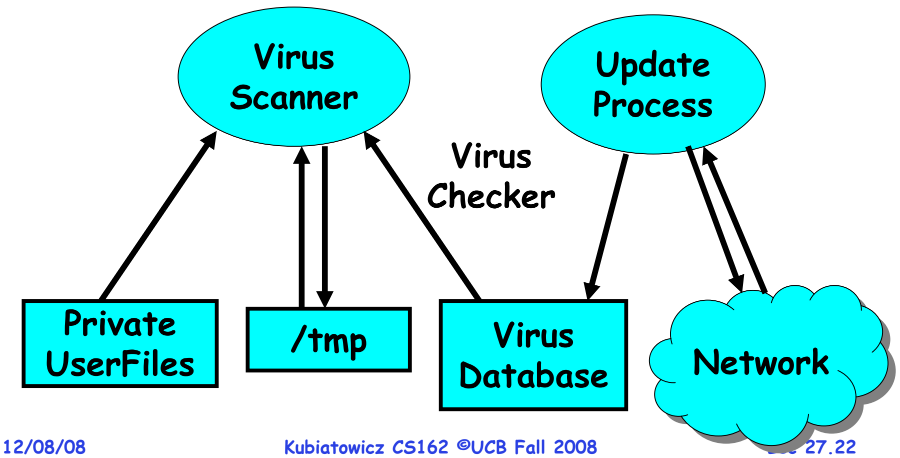
- Tenex used VM, and it interacts badly with the above code
  - Key idea: force page faults at inopportune times to break passwords quickly
- Arrange 1st char in string to be last char in pg, rest on next pg
  - Then arrange for pg with 1st char to be in memory, and rest to be on disk (e.g., ref lots of other pgs, then ref 1st page)

```
                a|aaaaaa
                 |
page in memory| page on disk
```

- Time password check to determine if first character is correct!
  - If fast, 1st char is wrong
  - If slow, 1st char is right, pg fault, one of the others wrong
  - So try all first characters, until one is slow
  - Repeat with first two characters in memory, rest on disk
- Only 256 * 8 attempts to crack passwords
  - Fix is easy, don't stop until you look at all the characters

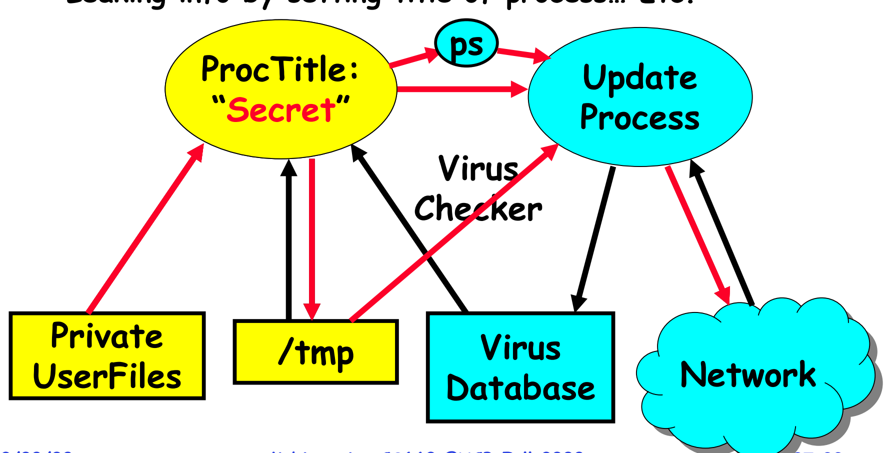
Protecting Information with Taint Tracking

- How can we prevent the illegal flow of information?
  - Consider Virus Scanner that scans your private files
    - » Example from Nickolai Zeldovich
  - What is to prevent a buggy scanner from leaking info?

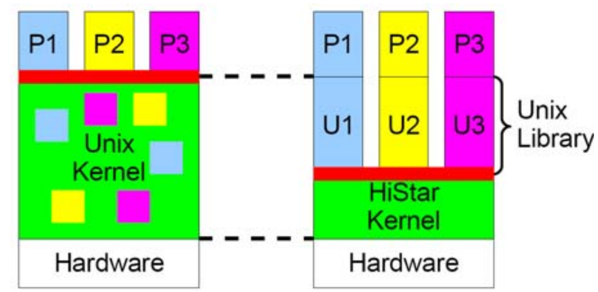


Possible avenues of leakage (MANY!)

- Possible ways of giving out private information:
  - Buggy Scanner gives out private info to update process
  - Leaks info through file system (or other file systems!)
  - Leaking info by setting title of process… Etc.



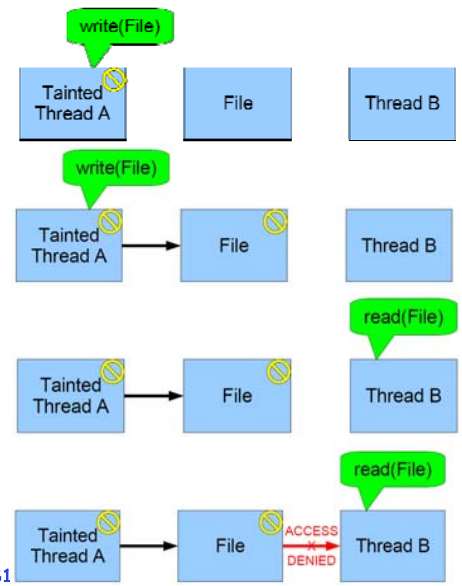
What is problem/Solution



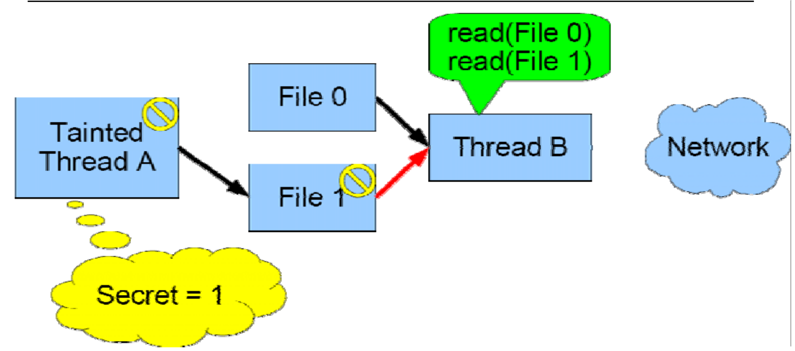
- Kernel not designed to enforce these policies
- Retrofitting difficult:
  - Must track any memory observed or modified by a system call!
  - Hard to even enumerate all possible channels
- Answer: Make all state explicit, track all communication
  - Example: Asbestos (MIT), HiStar (Stanford)
- Think of all data, threads, files, etc having a "Label"
  - Like a color; track colors through system, don't allow colors to "bleed" incorrectly into places they are not supposed to

Simple Taint Tracking Example

- Give a particular Label to every Thread
  - Propagate this label to all data modified by the thread
- Allow accesses only if accessing thread has a compatible Label
  - Deny access is labels do not match
- Question: Where do labels come from?
  - New Labels may be allocated dynamically by apps
  - No privileged "root"



Strawman has Covert Channel



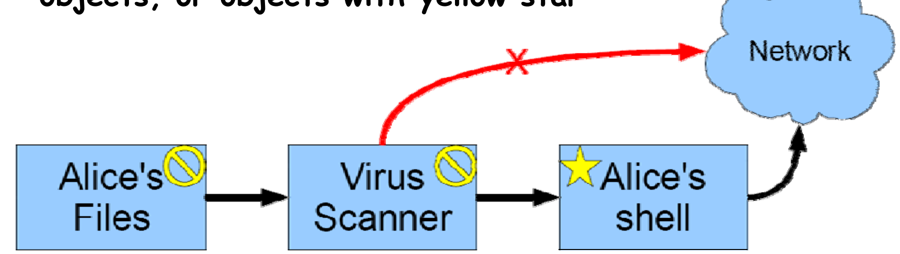
- Still possible to leak information by reflecting bits through failure
  - In example, Thread B finds out that secret is "1" because unable to read from File 1
- One fix to this covert channel: don't allow labels to change (i.e. must already exist, never propagated)
  - HiStar (Stanford) takes this approach

Asbestos Labels and Taint Tracking

- Labels are sets of pairs of (*category, Level*)
  - Category like "color" in previous examples
  - So, $L_x=\{ (h_1,l_1), (h_2,l_2), \ldots\ l_{default} \}$
    - » Notation: $L_x(a)$ = level of handle a in L_x or default
    - » They form a partial order: $L_1 \subseteq L_2$ if $\forall h$, $L_1(h) \leq L_2(h)$
  - Any active component of system can allocate new categories
    - » Could produce data that root cannot access
- Each entity (thread, file, socket,…) has send and receive label
  - Send level called "contamination".
    - » All outgoing messages tagged with send level of sender.
  - Receive level is max contamination allowed
- Communication from entity A to B allowed if $A_s \subseteq B_r$
  - After received, $B_s=B_s \cup A_s$
    - » Received message increases contamination level of receiving entity
  - Asbestos has special "*" level (the declassifier)
    - » Person with * in a category can declassify information tagged with that category and give it to anyone
    - » They can also read any information

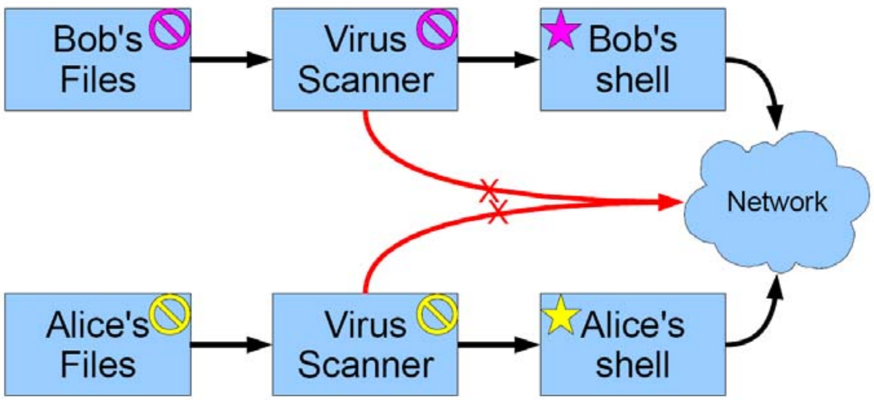
"Owner" privilege

- Yellow objects can only interact with other yellow objects, or objects with yellow star



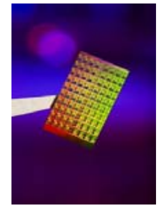
- Small, trusted shell can isolate a large, frequently-changing virus scanner
  - Try to reduce size of trusted code base
- Label checker is most trusted code and must be very carefully verified

Multiple categories of taint



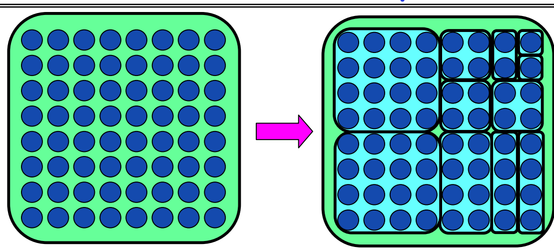
- Owner privilege and information flow control are the only access control mechanism
- Anyone can allocate a new category, gets star

ManyCore Chips: The future is here (for EVERYONE)



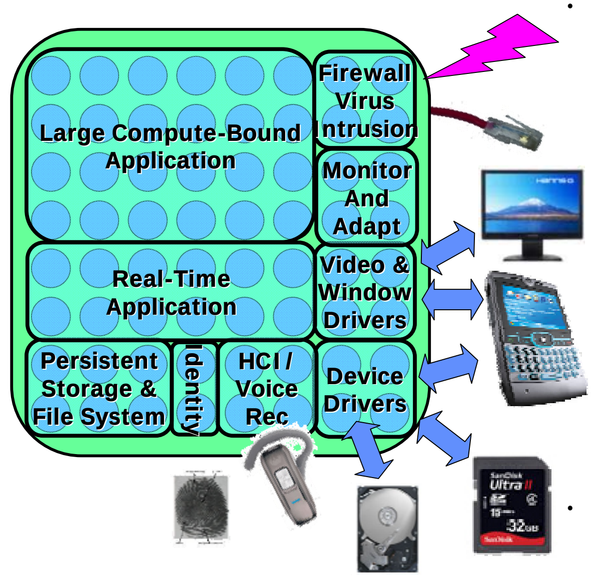
- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size
- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- Question: How can ManyCore change our view of OSs?
  - ManyCore is a challenge
    - » Need to be able to take advantage of parallelism
    - » Must utilize many processors somehow
  - ManyCore is an opportunity
    - » Manufacturers are desperate to figure out how to program
    - » Willing to change many things: hardware, software, etc.
  - Can we improve: security, responsiveness, programmability?

Important New Mechanism: Spatial Partitioning



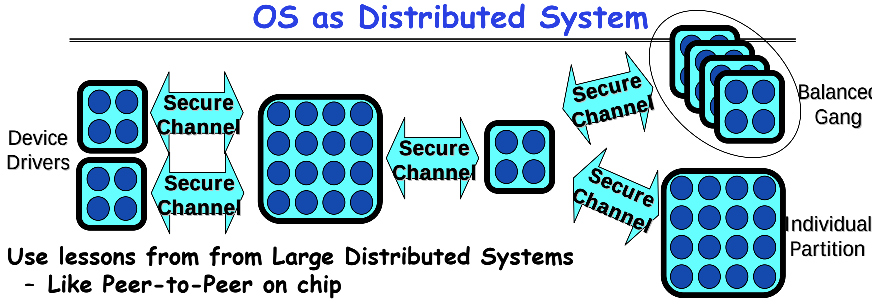
- Spatial Partition: group of processors acting within hardware boundary
  - Boundaries are "hard", communication between partitions controlled
  - Anything goes within partition
- Each Partition receives a *vector* of resources
  - Some number of dedicated processors
  - Some set of dedicated resources (exclusive access)
    - » Complete access to certain hardware devices
    - » Dedicated raw storage partition
  - Some guaranteed fraction of other resources (QoS guarantee):
    - » Memory bandwidth, Network bandwidth
    - » fractional services from other partitions
- Key Idea: Resource Isolation Between Partitions

Tessellation: The Exploded OS



- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - » TCP/IP stack
    - » Firewall
    - » Virus Checking
    - » Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - » Performance counters
    - » Introspection
  - Identity/Environment services (Security)
    - » Biometric, GPS, Possession Tracking
- Applications Given Larger Partitions
  - Freedom to use resources arbitrarily

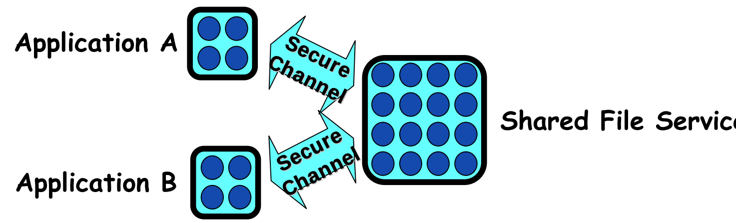
OS as Distributed System



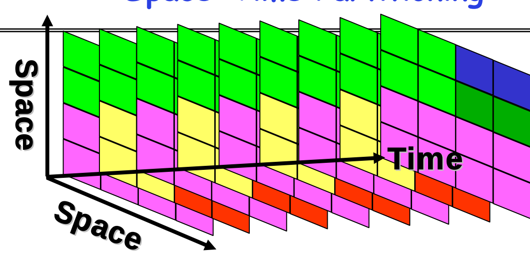
- Use lessons from from Large Distributed Systems
  - Like Peer-to-Peer on chip
  - OS is a set of independent interacting components
  - Shared state across components minimized
- Component-based design:
  - All applications designed with pieces from many sources
  - Requires composition: Performance, Interfaces, Security
- Spatial Partitioning Advantages:
  - Protection of computing resources *not required* within partition
    - » High walls between partitions ⇒ anything goes within partition
    - » "Bare Metal" access to hardware resources
  - Partitions exist simultaneously ⇒ fast communication between domains
    - » Applications split into distrusting partitions w/ controlled communication
    - » Hardware acceleration/tagging for fast secure messaging

It's all about the communication

- We are interested in communication for many reasons:
  - Communication represents a security vulnerability
  - Quality of Service (QoS) boils down message tracking
  - Communication efficiency impacts decomposability
- Shared components complicate resource isolation:
  - Need distributed mechanism for tracking and accounting of resource usage
    - » E.g.: How do we guarantee that each partition gets a guaranteed fraction of the service:



Space-Time Partitioning

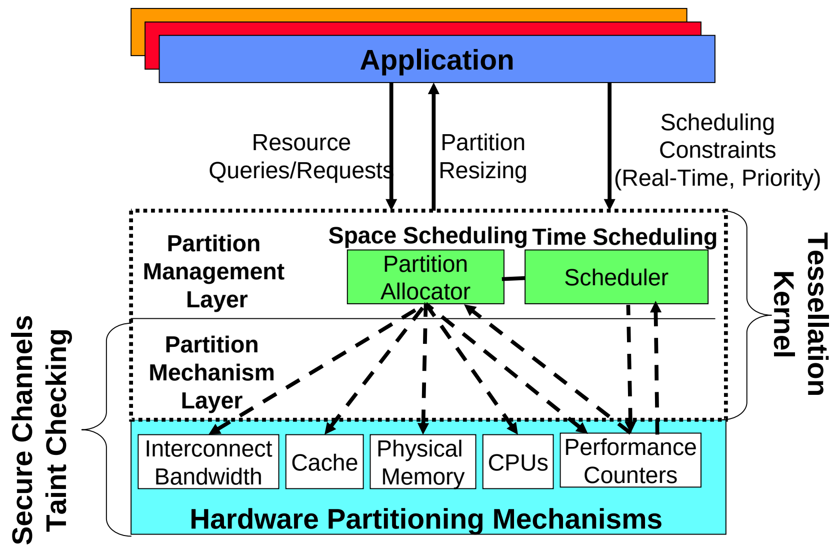


- Spatial Partitioning Varies over Time
  - Partitioning adapts to needs of the system
  - Some partitions persist, others change with time
  - Further, Partititions can be Time Multiplexed
    - » Services (i.e. file system), device drivers, hard realtime partitions
    - » Some user-level schedulers will time-multiplex threads within a partition
- Global Partitioning Goals:
  - Power-performance tradeoffs
  - Setup to achieve QoS and/or Responsiveness guarantees
  - Isolation of real-time partitions for better guarantees
- Monitoring and Adaptation
  - Integration of performance/power/efficiency counters

Another Look: Two-Level Scheduling

- First Level: Gross partitioning of resources
  - Goals: Power Budget, Overall Responsiveness/QoS, Security
  - Partitioning of CPUs, Memory, Interrupts, Devices, other resources
  - Constant for sufficient period of time to:
    - » Amortize cost of global decision making
    - » Allow time for partition-level scheduling to be effective
  - Hard boundaries ⇒ interference-free use of resources
- Second Level: Application-Specific Scheduling
  - Goals: Performance, Real-time Behavior, Responsiveness, Predictability
  - CPU scheduling tuned to specific applications
  - Resources distributed in application-specific fashion
  - External events (I/O, active messages, etc) deferrable as appropriate
- Justifications for two-level scheduling?
  - Global/cross-app decisions made by 1st level
    - » E.g. Save power by focusing I/O handling to smaller # of cores
  - App-scheduler (2nd level) better tuned to application
    - » Lower overhead/better match to app than global scheduler
    - » No global scheduler could handle all applications

Tessellation Partition Manager



Achieving Responsiveness & Agility

- Place time-critical components in their own partition
  - E.g.: User Interface Components, Jitter-critical applications
  - User-level scheduler tuned for deadline scheduling
- Grouping of external events to handle in next partition time slice
  - Achieving regularity (low standard deviation of behavior) more important than lowest latency for many types of real-time scheduling
  - Removes interrupt overhead (replaces it with polling)
- Pre-compose partition configurations
  - Quick start of partitions in response to I/O events or real-time triggers
- Judicious use of Speculation
  - Basic variant of the checkpointing mechanism to fork execution
  - When long-latency operations intervene, generate speculative partition
    - » Can track speculative state through different partitions/processes/etc
    - » Can be use to improve I/O speed, interaction with services, etc

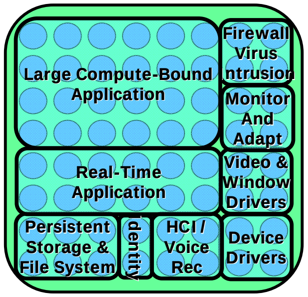
What about faults?

- Ignoring hardware and software failure is not an option!
  - Increased number of cores ⇒ increased failure rate
  - High software complexity because of parallelism
- Goal: Fast Restart of Partition after failed hardware or software
- Basic techniques: Checkpointing and Versioning with Detection
  - Providing automatic generation of stable restore points
    - » Periodic generation of checkpoints (basic)
    - » Framework (or application?) initiated checkpoints (more conservative)
  - Detecting when errors have occurred
    - » Low level errors (ECC, other failures)
    - » Framework-level checking of correctness signatures: still research topic
    - » Duplicate computation with online checking? (power intensive)
- Crash and Restart API to Productivity and Efficiency layers
  - Will allow application to say when to checkpoint and when to restart
- All centralized data structures versioned/transaction based?
  - Always possible to back out ("Undo") bad modification
  - Goal: allow components (such as device drivers) to crash and restart
  - File System (and "Object Storage") versioned

Conclusion

- Distributed identity
  - Use cryptography (Public Key, Signed by PKI)
- Use of Public Key Encryption to get Session Key
  - Can send encrypted random values to server, now share secret with server
  - Used in SSL, for instance
- Authorization
  - Abstract table of users (or domains) vs permissions
  - Implemented either as access-control list or capability list
- Issues with distributed storage example
  - Revocation: How to remove permissions from someone?
  - Integrity: How to know whether data is valid
  - Freshness: How to know whether data is recent
- Buffer-Overrun Attack: exploit bug to execute code
- Taint Tracking
  - Track flow of information
  - Protect data rather than processes

Conclusion (Con't)



- ManyCore: the future is here!
- Tessellation Goals: RAPPidS
  - Responsiveness, Agility, Power-Efficiency, Persistence, Security
  - User experience, real-time behavior, efficient use of resources
- Spatial Partitioning: grouping processors & resources behind hardware boundary
  - Two-level scheduling
    1) Global Distribution of resources
    2) Application-Specific scheduling of resources
  - Bare Metal Execution within partition
  - Composable performance, security, QoS
- Tessellation OS
  - Exploded OS: spatially partitioned, interacting services