Software Safety
Case Studies

• Therac-25 medical accelerator, 1987
• National Cancer Institute, 2000
• The 1\textsuperscript{st} worm virus, 1988
• 1994 Chinook helicopter crashes
• What really happened on Mars? 1997
• IE in my computer does not go back, now
Common Causes

• arithmetic bugs
  - division by zero, overflow, ..
  - precision
• logic bugs
• syntax bugs
• resource bugs
  - null pointer, uninitialized variables
  - access violations
  - buffer(stack) overflow, memory leak
• multi-threading bugs
  - with multicore, .....
Safety from what?

• External invasion (hacking)
  - security
  - it is another big area of computer science

• Internal flaw (protection)
  - a bug in a software module
  - OS is protected from a bug in user applications
  - then, why are we worried about OS safety?
    • unreliable OS code itself
    • 3rd party device drivers
    • OS extensions
Basic Techniques

- In PL, static analysis is favored
  - accurate but small coverage
  - a good programming language incurs less bugs
- In SE, the whole lifecycle is considered
  - the spectrum is too big to handle properly
  - they think faults come from bad design, not from a bad guy
- In our world,
  - belief: a bug will always be out there
  - tolerate faults
  - isolate faults
  - restart from a safe state (in the hope that the bug disappears next time)
Security Overview
Protection Overview

• **Security**
  - protect the system from external attacks
  - Trojan horses(virus), covert channel, worm, ....

• **Protection**
  - protect the integrity of computer system from intentional/unintentional attacks from outside/inside
  - confine the access privilege of a process
Access Matrix

- **Domain Structure**
  
  domain A
  
  - \(<O_1, \{\text{read, write}\}>\)
  - \(<O_2, \{\text{read, write}\}>\)
  - \(<O_3, \{\text{execute}\}>\)
  - \(<O_4, \{\text{read, write}\}>\)

  domain B
  
  - \(<O_1, \{\text{read, write}\}>\)
  
  - \(<O_4, \{\text{read, write}\}>\)

  domain C
  
  - \(<O_3, \{\text{execute}\}>\)
  
  a domain can be a process, user, ..
Access Matrix

- Provides the mechanism for various policies
- Problematic operations
  - a process creates a new file
  - a process forks a new child process
  - I want to share a file with Susan, but do not want Susan to share it with others
  - ...
  - the matrix is huge and sparse

<table>
<thead>
<tr>
<th>domain</th>
<th>file 1</th>
<th>file 2</th>
<th>file 3</th>
<th>file 4</th>
<th>. . . .</th>
<th>printer</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>r</td>
<td>w</td>
<td></td>
<td></td>
<td>. . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td>print</td>
</tr>
<tr>
<td>D3</td>
<td>r</td>
<td>w</td>
<td></td>
<td></td>
<td>. . . .</td>
<td></td>
<td>enter</td>
</tr>
<tr>
<td>D4</td>
<td>w</td>
<td>w</td>
<td></td>
<td>r</td>
<td>. . . .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementations

- **ACL (Access Control List):** a list for each object
  - 

- **Capability list:** a list for each domain
  - 

<table>
<thead>
<tr>
<th>domain</th>
<th>file 1</th>
<th>file 2</th>
<th>file 3</th>
<th>file 4</th>
<th>...</th>
<th>printer</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>r</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>r</td>
<td></td>
<td>w</td>
<td></td>
<td></td>
<td>enter</td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>w</td>
<td>w</td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Capability

• need a mechanism to protect capabilities
  - user just present the address of a capability, not the content of it to the system
  - if the capability contents is in user space, it needs protection
    • tagged memory
    • partitioned memory

• advantages
  - simple
    • owning a capability means the right of access
  - efficient
    • easy to test the validity of access
  - flexible
    • transfer, sharing
Capability(2)

• revocation
  - suppose
    • X gives a capability to Y
    • Y stores it in a hidden data structure
    • X wants to revoke the capability
  - destroy the object and make a copy of object
    • the whole capabilities should be reconstructed

• garbage collection
  - when all the capabilities to an object disappear, the object becomes garbage
  - the creator maintains the count of capabilities
    • should track the number of copies made
ACL

• advantages
  - easy revocation and review
  - easy control of propagation
• disadvantages
  - slow
    • for every access to an object the whole list should be examined (this list may be quite long)
    • cacheing of ACL in the subject’s space may help but
      - it is a capability, and all the problems of it pop up
  - storage
    • capabilities are stored in each process’s space
    • ACL is usually stored in system space
    • users may be grouped
    • default values can be helpful
Safe State

• a strong definition
  a process cannot acquire an access right to a file without the consent of its owner
  - impractical since
    • a right should be conferrable
    • a file should be sharable by anonymous processes

• a weaker definition
  a process should be able to tell whether its actions can lead to the leakage of an access right to unauthorized subjects
  - still strong since
    • it is undecidable whether a sequence of commands may lead the leakage
Take–Grant Model

- a directed graph version of the access control matrix
  - node: subject or object
  - labels
    - read, write, execute
    - take: x can take rights of y
    - grant: y can be granted rights of x
Bell–La Padula Model

- the model (for DoD)
  - several ordered security levels
  - each subject has a clearance
  - each object has a classification
- accesses
  - read-only
  - append: write without read privilege
  - execute: without read/write privileges
  - read-write
- control attribute
  - associated with each object
  - defines who can control the access privileges on this object
  - access rights can be transferred but control attribute cannot be transferred
Bell–LaPadula Model (2)

• current security clearance level
  - a clearance level of a subject at an instance
  - objects whose classification is the same as CSCL
    • can be read, modified
  - objects whose classification is lower than CSCL
    • can be read (reading down property)
  - objects whose classification is higher than CSCL
    • can be appended (writing up property)
• The \textbf{*-property} \\
  - the \textbf{reading down} property prevents accesses to an object at a higher level \\
  - the \textbf{writing up} property prevents releasing information to other subjects via objects at lower level \\
  \hspace{1em} • you can send information only upward, NOT downward \\
  \hspace{1em} • but downward flow is sometimes needed
Lattice Model

• lattice
  - a mathematical structure where elements are ordered under a partial ordering
  - single source and single sink
    • these conditions can be met by dummy elements

• an object $x$ belongs to a security class $x$

• information flow $x \rightarrow y$
  - information flow from $x$ to $y$ is permitted
  - reflexive: information flow within a class is permitted
  - antisymmetric
  - transitive
Lattice Model (2)

• advantages
  - wide range of policies can be modeled
  - simple definition of safety
    • if the execution of a process does not result in an information flow from $x$ to $y$ unless $x \rightarrow y$

• example: military security model
  - objects are ranked
    • unclassified
    • confidential
    • secret
    • top secret
  - a class of an object: (rank, compartments)
  - a clearance of a subject: (rank, compartment)
  - access is allowed only when
    • rank of subject $\geq$ rank of object
    • compartment of subject $\subseteq$ compartments defined on the object
OS Functions for Security

- authentication of users
- protection of memory
- file and device access control
- allocation and access control to general object
- enforcement of sharing
  - concurrency control
  - transaction support
- guarantee of fair service
  - a process should not be able to monopolize system resources
- IPC and synchronization
  - IPC should be mediated by access control tables
- audit
- intrusion detection
Authentication

• Security
  - prevent intentional misuse
  - three pieces
    • authentication: who are you?
    • authorization: who is allowed to do what
    • enforcement: make sure people do only what they are allowed to do

• Authentication
  - common approach: password
  - Can we trust encryption algorithm?
    • if there is backdoor open, ...
  - public key + digital signature
Authentication(2)

• Private key encryption
  - key should be protected securely
  - DES (Data Encryption Standard)
• Public key encryption
  - alternative to private key
  - each key is a pair of $K$, $K^{-1}$
    - keep $K$ private, and $K^{-1}$ in public
  - with private key (text)$KK = \text{text}$
  - with public key (text)$KK^{-1} = \text{text}$
  - (I am Joon)$^K$ : everyone can read it, but only I can send it
    - signature
  - (Hi!)$^{K^{-1}}$ : anyone can send it, but only I can read it
  - RSA algorithm
Kerberos

- two parties, A, B trust a server, but NOT each other
- Notation
  • $K_{xy}$ is a key used between x and y
  • $(..)^K$ is an encrypted message with a key $K$
- Operation
  • A asks server for key: A $\rightarrow$ S (Hi, I need a key, Kab)
  • Server gives back a session key encrypted in B's key
    - (Use Kab; (This is A! Use Kab)${}^K_{s_b}$)${}^K_{s_a}$
  • A sends B
    - (This is A! Use Kab)${}^K_{s_b}$ ----(1)
- Intruder can record and resend the message (1) to B
  • B replies to A a counter
  • A replies back to B (counter-1)
- Need encrypted checksum in case malicious user inserts garbage into the message
Digital Signature

- a sender $A$ computes characteristic value $CS$ for the message
  - $CS$ must be unique to each message
- $A$ sends the server $(CS)^{K_a}$
- server generates a signature $S$ based on $CS$
- the server sends $A (S)^{K_s}$ ---- $K_s$ is private key of the server
- $A$ sends $(\text{message, } CS, (S)^{K_s})^{K_a}$ to $B$
- $B$ sends $(S)^{K_s}$ to the server for decrypting $S$
  - it is already encrypted
- the server computes $CS'$ from $S$
- the server sends $B (CS')^{K_b}$
- $B$ checks if $CS = CS'$
Recovery
Introduction

• failure
  - when a system does not perform in the manner defined
• erroneous state
  - state that could lead the system to the failure
• fault
  - anomalous physical condition
  - causes
    • design/manufacturing error
    • damage/fatigue
    • external disturbance
• faults lead the system to an erroneous state which may or may not results in a failure
Failures

• process failure
  - deadlock, timeout, protection violation, ...
  - OS should confine this failure to the process

• system failure
  - software and hardware
  - amnesia failure: cannot recover the state just before the failure
  - pause failure: the state can be reinstated
  - halting failure: the system never restarts

• disk failure
  - serious problem when it is the last backup storage
  - usually backed up by tape OR
  - mirrored (it will enhance read throughput anyway)

• communication medium failure
  - does not cause total system failure
Error Recovery

• **Forward Error Recovery**
  - allow the process to proceed after fixing errors
  - difficult to remove all the errors (in software, procedures to cope with all kinds of error should be prepared, which is almost impossible)

• **Backward Error Recovery**
  - the process should restart from the saved (or predefined) state
  - roll-back mechanism is needed
  - easy to cope with any kind of errors (it is not necessary to anticipate all kinds of errors)
  - overhead to restore previous state
    - checkpointing is needed
  - same errors may occur again
Backward Error Recovery

• Operation-based approach
  - using a log, undo(roll-back) what has been done until an error-free state can be restored
  - write ahead log (for a write to X)
    • records in a log new value of X
    • updates X

• State-based approach
  - checkpoint
    • a complete state of a process
    • at crash, rollback to the most recent safe state
      - needs many checkpoints
  - shadow page
    • copy of a page that is to be updated
    • updates are done only on the original page
    • at crash, goes back to the shadow page
    • at commit, keep using the original page
Issues in Recovery (1)

- failure and recovery of a process affect other processes that exchange data with the failed process
- orphan message
  - when a process rolls back to the point before sending out a message
  - actions of other processes depending on the orphan message should be rolled back, too (domino effect)
- lost message
  - node Y receives a message from X
  - Y rolls back to the point before receiving the message
  - effects are the same as when the message is lost
Issues in Recovery(2)

• **livelocks**

  1. failure, and roll back
  2. orphan message, roll back

- Y sends out m1 and receives an orphan message n1, and rolls back
- m1 becomes an orphan message
- receiving m1, X rolls back
Recoverable Virtual Memory

- cope with process failure
- virtual address space for transactions
  - space for transactions is declared by program
  - the space is copied to the disk
  - modifications are performed on original pages
  - abort means restoring the disk copy
- no-undo/redo with logging
  - write ahead log
  - at commit the log should be in a stable storage
    - this can be delayed by applications (no-flush transactions) such as ones that read only
  - no undo is needed because of disk copy
Rio Vista

• crash taxonomy
  - hardware: not frequent
  - software: frequent due to bugs in OS
  - power: UPS

• motivations
  - transactions are useful but high overhead (disk accesses)
  - file cache is useful, but vulnerable to system crashes
Reliable File Cache

• protect cached data from system crashes
  - cache is as reliable as a disk
  - then, write ahead log for recovery is not needed
  - writes to disk can be delayed infinitely

• OS errors can corrupt any part of the system
  - the issue is how to reduce the chances

• at a crash
  - warm reboot process writes the cache to disk
File Cache vs Disk

• why people view memory more vulnerable than disk?
• memory access is a simple write
  - an error in the address bits will overwrite the file cache
• interface to access disk is complex and explicit
  - hardware controller is accessed only through device driver
  - calls to device drivers are checked for their arguments
  - it is extremely unlikely that accidental errors can forge the logic of device driver
How to protect from system crashes?

• prevent OS from accidentally overwriting the file cache

• virtual memory mapping
  - turn off the write-permission bits in the page table for the pages in the file cache
  - unauthorized accesses will encounter protection violation
  - file cache module enables the bit before writing and disables the bit afterwards
  - the file cache is vulnerable to crashes while being written
    • verify after writes
    • use shadow copy for atomic writes
• some kernels bypass the address translations (TLB)
  - many systems can disable such bypasses
  - otherwise, code insertion (sandboxing)
    • check for every kernel write using physical address
    • 20-50% slower
• memory-mapped file
  - kernel procedures that modify the memory-mapped file should be changed as above
  - faulty user program can still corrupt files to which it has write access
Warm Reboot

- Recovery needs to access many data structures
  - internal file cache lists
  - page tables (memory-mapped files)
  - all these data must be protected from crash but they are scattered inside the kernel
- Registry
  - a separate physical memory region
  - contains all the information to recover the file cache
  - it is updated only when a buffer is replaced (reloaded)
- effects on file system
  - writes to disk can be saved
    - most disk writes are reliability-induced
  - writes to disk are needed only when the file cache overflows
  - writing back dirty copies when the system is idle
    - reduces the time when a buffer is replaced
Persistent Heap

- transactions may use
  - when they abort, all the used heaps are returned
- undo records are stored here (why?)
- programs can store their original data structures
  - usually convert them to record style when stored in a file
- meta data for the heap is in user space
  - need a protection from corruption
    - reduce the risk by using isolated range of addresses
    - software fault isolation
    - virtual memory protection
OS Extension
OS Extensions

• Loadable Kernel Module (LKM)
  - OS functions are defined at compile time
  - available for most OS’es
    • Unix, Linux, MS Windows, Mac OS X

• OS can be tainted
• fragmentation may occur
• security can be compromised
Graft

- **graft**
  - user code inserted into the kernel
  - can corrupt the kernel
    - buggy graft
    - malicious graft
- **needed kernel protection from graft**
  - memory protection
    - read/write inappropriate data
    - execution of bad instructions
  - resource protection
    - monopolizing may degrade system performance and jeopardize the acceptable performance of applications
Graft (2)

• **graft model**
  - protect kernel by SFI
  - protect resources by using transactions
    • kernel logs graft actions to undo if the grafts misbehave
• **why a graft is dangerous**
  - it runs in the kernel mode
    • can corrupt any part of the system
  - most kernel interfaces are open to it
    • most procedure interfaces
    • unlike well-defined syscall interface
    • we can restrict the interfaces open to a graft
      - tradeoff between safety and graft functionality
  - system relies on a graft
Graft Misbehaviors

- illegal data access
  - VM mechanism
    - in kernel mode, a graft can bypass address translation
  - use of safe language or SFI
    - need to verify that a graft is made by such a tool
  - kernel shouldn’t give any information to grafts
    - maybe, metadata only

- resource hoarding
  - resources of interest
    - CPU, memory, network, synchronization
  - treat a graft like a user level process
    - preempt a graft for a sharable resource
    - terminate a graft for an exclusive resource (needs rollback)
Graft Misbehaviors(2)

• access to incorrect interface (that are not allowed to access)
  - to change a global policy
    • scheduling, memory allocation
  - to shutdown
  - to access private data

• antisocial behavior
  - a graft does not do what it has agreed to do
  - when a (scheduling) graft affects only the members of the group, it is acceptable
  - a graft should not affect the processes in other group

• covert denial of service
  - when a kernel asks a graft of a decision, the graft may never returns
    • the kernel cannot make any progress
  - timeout mechanism is needed
Restrictions on Grafts

- Grafts must be preemptible.
- Grafts cannot hold kernel locks or limited kernel resource for excessive periods of time.
- Grafts cannot access memory to which they have not been granted permission.
- Grafts cannot call functions that alter or return data that the graft is not allowed to access.
- Grafts cannot replace restricted kernel functions.
Restrictions on Grafts (2)

- the kernel must not execute grafts that are not known to be safe
- grafts must not call functions to which they have not been granted access
- malicious grafts can only affect applications that have agreed to use them
- the kernel must be able to make progress even with a faulty graft in its path
Grafting Architecture

• an extension is a transaction
  - can be undone

• modes of extension
  - replacing a method on an object
    • overrides default policy
  - handler for a given event
    • new kernel level services
    • spawns a new thread
Kernel Transaction Support

- when a graft is inserted, a wrapper is interposed
  - it begins a transaction and calls the graft
  - when the graft returns to the wrapper without any problem, the wrapper commits the transaction
- redo log is not necessary
- when a transaction must be aborted,
  - transaction manager invokes undo operations
    - using the undo call stack
  - returns error to the graft stub
  - invoke default function
- nested transactions
  - when a transaction aborts, it just returns abort state to the parent transaction
  - when it commits, the undo call stack is merged with parent’s
Aborting a Graft

- **time-constrained resources**
  - when a graft is holding it for too long, other processes cannot make progress
    - lock
  - for every lockable resource, a timeout value is assigned
    - the value depends on the type of resource
  - when a thread blocks on such a resource, it schedules a time-out
    - when it goes off, the lock holding thread is aborted

- **quantity-constrained resources**
  - a graft should not use it too much
    - memory, buffer, ....
  - each graft thread is given a limit for each resource
    - these limits are assigned when a graft is installed
    - ticket delegation in **lottery scheduling** can be used
Graft Code Safety

• a list of graft-callable functions is defined by the OS developer
• a graft is checked if it makes legitimate calls
• indirect function call is checked at run time
  – by inspecting (by hashing) if it is on the list
• graft-callable functions performs the same level arguments checking as for syscalls
• how to verify that a graft comes through SFI tool?
  – digital signature
Installing a graft

- **a method graft**
  - kernel has predefined graft points: (ob, method)
  - a user process invokes a syscall for installing a graft
    - install_graft(function, graft_point)
  - kernel finds the address of the function to be replaced
  - get a space for the graft
  - assign the address of graft to the calling instruction
Installing a graft(2)

• an event graft
  - mechanism
    • kernel defines only the event types to be processed
    • user defines new graft points to the kernel
      - using syscall
    • user installs a graft at the graft point
  - difference from the method graft
    • it is a separate thread
    • method applies only to an object while an event can be applied to the whole kernel
    • a new graft can be added without replacing old one
Other OS Extensions

• **SPIN**
  - extensions are even handlers written in Modula-3
  - examples: network protocol, web server, ..

• **Fox with PCC**
  - powerful, but I cannot generate a proof

• **Exokernel**