Formal Models for Computer Security

- Two fundamental computer security facts:
  - all complex software systems have flaws/bugs
  - is extraordinarily difficult to build computer hardware/software not vulnerable to attack
- Hence, the desire to prove design and implementation satisfy security requirements
- Led to development of formal security models
  - initially funded by US DoD
  - Bell-LaPadula (BLP) model very influential

Bell-LaPadula (BLP) Model

- developed in 1970s
- as a formal access control model
- subjects and objects have a security class
  - top secret > secret > confidential > unclassified
  - subject has a security clearance level
  - object has a security classification level
  - class controls how subject may access an object

BLP Formal Description

- based on current state of system \((b, M, f, H)\):
  \((current\ access\ set\ b,\ access\ matrix\ M,\ level\ function\ f,\ hierarchy\ H)\)
- three BLP properties:
  - ss-property: \((S_i, O_j, \text{read})\) has \(f(S_i) \geq f(O_j)\).
  - *-property: \((S_i, O_j, \text{append})\) has \(f(S_i) \leq f(O_j)\) and \((S_i, O_j, \text{write})\) has \(f(S_i) = f(O_j)\).
  - ds-property: \((S_i, O_j, A_i)\) implies \(A_i \in M[S_i, O_j]\)
- BLP give formal theorems
  - theoretically possible to prove system is secure
  - in practice usually not possible

BLP Abstract Operations

1. get access
2. release access
3. change object level
4. change current level
5. give access permission
6. rescind access permission
7. create an object
8. delete a group of objects
**Integrity Policy Models**

- **Requirements:**
  - Very different from confidentiality policies
  - We are describing whether (and how much) we can trust data. (Not who can see it.)
- **Integrity:** the state that exists when records agree with the source from which they were derived and have not been incorrectly altered or destroyed.
- Biba’s model
- Clark-Wilson model

**Principles**

- **Separation of duties:**
  - Two or more people are required to perform critical functions. (May require separation into two or more steps)
- **Separation of function:**
  - Development separated from production, etc.
- **Auditing:**
  - It must be possible to reconstruct what happened, by whom, and when.

**Intuition for Integrity Levels**

- The higher the level, the more confidence
- That a program will execute correctly
- That data is accurate and/or reliable
- Note: there is a relationship between integrity and trustworthiness
- Important point: integrity levels are not security levels. Integrity is concerned with trustworthiness, not data flow.

**Biba Integrity Model**

- Set of subjects $S$, objects $O$, integrity levels $I$.
- $\leq$ on a subset of $I \times I$ holds when the second level dominates or is the same as the first.
- $\min: I \times I \to I$ returns lesser of integrity levels
- $i: S \cup O \to I$ gives integrity level of entity
- $r: S \times O$ means $s \in S$ can read $o \in O$
- $w, x$ (write, execute) defined similarly
- This is a dual of Bell-LaPadula

**Biba’s Model**

- Similar to Bell-LaPadula model
  1. $s \in S$ can read $o \in O$ iff $i(s) \leq i(o)$
  2. $s \in S$ can write to $o \in O$ iff $i(o) \leq i(s)$
  3. $s_r \in S$ can execute $s_x \in S$ iff $i(s_x) \leq i(s_r)$
- Add compartments to get full dual of Bell-LaPadula model
- This is actually the “strict integrity model” of Biba’s set of models

**Biba Integrity Model**

- various models dealing with integrity
- **strict integrity policy:**
  - simple integrity: $I(S) \geq I(O)$
  - integrity confinement: $I(S) \leq I(O)$
  - invocation property: $I(S_1) \geq I(S_2)$
Clark-Wilson Integrity Model

- Integrity defined by a set of constraints
  - Data in a consistent or valid state when it satisfies constraints.
- Example: Bank
  - \( D \) today’s deposits, \( W \) withdrawals, \( YB \) yesterday’s balance, \( TB \) today’s balance
  - Integrity constraint: \( YB + D - W = TB \)
- Well-formed transactions move system from one consistent state to another
- Issue: who examines, certifies transactions done correctly?

Entities

- CDIs: constrained data items
  - Data subject to integrity controls
- UDIs: unconstrained data items
  - Data not subject to integrity controls
- IVPs: integrity verification procedures
  - Procedures that test the CDIs conform to the integrity constraints
- TPs: transaction procedures
  - Procedures that take the system from one valid state to another

Comparison to Biba

- Biba
  - No notion of certification rules; trusted subjects ensure actions obey rules
  - Untrusted data examined before being made trusted.
- Clark-Wilson
  - Explicit requirements that actions must meet
  - Trusted entity must certify method to upgrade untrusted data (and not certify the data itself)

Chinese Wall Model

Problem:
- Tony advises Bank of America about investments
- He is asked to advise CitiBank about investments
- Conflict of interest to accept, because his advice for either bank would affect his advice to the other bank

Organization

- Organize entities into “conflict of interest” classes
- Control subject accesses to each class
- Control writing to all classes to ensure information is not passed along in violation of rules
- Allow sanitized (public) data to be viewed by everyone

Definitions

- Objects: items of information related to a company
- Company dataset (CD): contains objects related to a single company
  - Written \( CD(O) \)
- Conflict of interest class (COI): contains datasets of companies in competition
  - Written \( COI(O) \)
- Assumption: each object belongs to exactly one COI class
**Example**

Bank COI Class
- Bank of America
- Citibank
- Bank of the West

Gasoline Company COI Class
- Shell Oil
- Standard Oil
- Union ‘76
- ARCO

**Temporal Element**
- If Anthony reads any CD in a COI, he can *never* read another CD in that COI
- Possible that information learned earlier may allow him to make decisions later
- Let $PR(S)$ be set of objects that $S$ has already read

**CW-Simple Security Condition**
- $s$ can read $o$ iff either condition holds:
  1. There is an $o'$ such that $s$ has accessed $o'$ and $CD(o) = CD(o')$.
     Meaning $s$ has already read something in $o$’s dataset
  2. For all $o' \in O$, $o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$
     Meaning $s$ has not read any objects in $o$’s conflict of interest class
- Ignores sanitized data
- Initially, $PR(s) = \emptyset$, so initial read request granted

**Sanitizing**
- Public information may belong to a CD
  - As it’s publicly available, no conflicts of interest arise
  - So, should not affect ability of analysts to read
  - Typically, all sensitive data removed from such information before it is released publicly (called sanitizing)
- Add third condition to CW-Simple Security Condition:
  3. $o$ is a sanitized object

**Writing**
- Anthony, Susan work in same trading house
- Anthony can read Bank 1’s CD, Gas’ CD
- Susan can read Bank 2’s CD, Gas’ CD
- If Anthony could write to Gas’ CD, Susan can read it
  - Hence, indirectly, she can read information from Bank 1’s CD, a conflict of interest

**CW-*-Property**
- $s$ can write to $o$ iff both of the following hold:
  1. The CW-simple security condition permits $s$ to read $o$; and
  2. For all *unsanitized* objects $o'$, if $s$ can read $o'$, then $CD(o') = CD(o)$
- Says that $s$ can write to an object if all the (unsanitized) objects it can read are in the same dataset
**Compare to Clark-Wilson**

- Clark-Wilson Model covers integrity, so consider only access control aspects
- If “subjects” and “processes” are interchangeable, a single person could use multiple processes to violate CW-simple security condition
  - Would still comply with Clark-Wilson Model
- If “subject” is a specific person and includes all processes the subject executes, then consistent with Clark-Wilson Model

**ORCON**

- Problem: organization creating document wants to control its dissemination
- Example: Secretary of Agriculture writes a memo for distribution to her immediate subordinates, and she must give permission for it to be disseminated further. This is “originator controlled” (here, the “originator” is a person).

**Requirements**

- Subject \( s \in S \) marks object \( o \in O \) as ORCON on behalf of organization \( X \). \( X \) allows \( o \) to be disclosed to subjects acting on behalf of organization \( Y \) with the following restrictions:
  1. \( o \) cannot be released to subjects acting on behalf of other organizations without \( X \)’s permission;
  2. Any copies of \( o \) must have the same restrictions placed on it.

**DAC Fails**

- Owner can set any desired permissions
- This makes 2 (application of ORCON rules to copies) unenforceable

**MAC Fails**

- First problem: category explosion
  - Category \( C \) contains \( o, X, Y \), and nothing else. If a subject \( y \in Y \) wants to read \( o, x \in X \) makes a copy \( o' \). Note \( o \) has category \( C \). \( y \) wants to give \( z \in Z \) a copy, \( z \) must be in \( Y \)—by definition, it’s not. If \( x \) wants to let \( w \in W \) see the document, need a new category \( C' \) containing \( o, X, W \).
- Second problem: abstraction
  - MAC classification, categories centrally controlled, and access controlled by a centralized policy
  - ORCON controlled locally

**Combine Them**

- The owner of an object cannot change the access controls of the object.
- When an object is copied, the access control restrictions of that source are copied and bound to the target of the copy.
  - These are MAC (owner can’t control them)
- The creator (originator) can alter the access control restrictions on a per-subject and per-object basis.
  - This is DAC (owner can control it)
RBAC

- Access depends on function, not identity
  - Example:
    - Allison, bookkeeper for Math Dept, has access to financial records.
    - She leaves.
    - Betty hired as the new bookkeeper, so she now has access to those records
  - The role of “bookkeeper” dictates access, not the identity of the individual.

Multilevel Security (MLS)

- A class of system that has system resources (particularly stored information) at more than one security level (i.e., has different types of sensitive resources) and that permits concurrent access by users who differ in security clearance and need-to-know, but is able to prevent each user from accessing resources for which the user lacks authorization.

MLS Security for Role-Based Access Control

- Role-based access control (RBAC) can implement BLP MLS rules given:
  - Security constraints on users
  - Constraints on read/write permissions
  - Read and write level role access definitions
  - Constraint on user-role assignments

Reference Monitors

- Complete mediation
- Isolation
- Verifiability

Trojan Horse Defence

Multilevel Security (MLS)

- A class of system that has system resources (particularly stored information) at more than one security level (i.e., has different types of sensitive resources) and that permits concurrent access by users who differ in security clearance and need-to-know, but is able to prevent each user from accessing resources for which the user lacks authorization.
MLS Security for Role-Based Access Control

- rule based access control (RBAC) can implement BLP MLS rules given:
  - security constraints on users
  - constraints on read/write permissions
  - read and write level role access definitions
  - constraint on user-role assignments

MLS Database Security

Read Access
- DBMS enforces simple security rule (no read up)
- easy if granularity entire database / table level
- inference problems if there is column granularity
  - if can query on restricted data can infer its existence
    - SELECT Ename FROM Employee WHERE Salary > 50K
  - solution is to check access to all query data
- also have problems if have row granularity
  - null response indicates restricted/empty result
- no extra concerns if have element granularity

Write Access
- enforce *-security rule (no write down)
- have problem if a low clearance user wants to insert a row with a primary key that already exists in a higher level row:
  - can reject, but user knows row exists
  - can replace, compromises data integrity
  - can polyinstantiate and insert multiple rows with same key, creates conflicting entries
- same alternatives occur on update
- avoid problem if use database / table granularity

Trusted Platform Module (TPM)
- concept from Trusted Computing Group
- hardware module at heart of hardware / software approach to trusted computing
- uses a TPM chip on
  - motherboard, smart card, processor
- working with approved hardware / software
- generating and using crypto keys
- has 3 basic services: authenticated boot, certification, and encryption
Authenticated Boot Service

- responsible for booting entire O/S in stages
- ensuring each is valid and approved for use
  - verifying digital signature associated with code
  - keeping a tamper-evident log
- log records versions of all code running
- can then expand trust boundary
  - TPM verifies any additional software requested
    - confirms signed and not revoked
- hence know resulting configuration is well-defined with approved components

Encryption Service

- encrypts data so it can be decrypted
  - by a certain machine in given configuration
- depends on
  - master secret key unique to machine
  - used to generate secret encryption key for every possible configuration only usable in it
- can also extend this scheme upward
  - create application key for desired application version running on desired system version

TPM Functions

Protected Storage Function

Trusted Systems

- security models aimed at enhancing trust
- work started in early 1970’s leading to:
  - Trusted Computer System Evaluation Criteria (TCSEC), Orange Book, in early 1980s
  - further work by other countries
  - resulting in Common Criteria in late 1990s
- also Computer Security Center in NSA
  - with Commercial Product Evaluation Program
    - evaluates commercially available products
    - required for Defense use, freely published

Common Criteria (CC)

- ISO standards for security requirements and defining evaluation criteria to give:
  - greater confidence in IT product security
  - from formal actions during process of:
    - development using secure requirements
    - evaluation confirming meets requirements
    - operation in accordance with requirements
  - evaluated products are listed for use
CC Requirements
• have a common set of potential security requirements for use in evaluation
• target of evaluation (TOE) refers to product/system subject to evaluation
• functional requirements
  • define desired security behavior
• assurance requirements
  • that security measures are effective and correct
• have classes of families of components

Smartcard PP
• simple Protection Profile example
• describes IT security requirements for smart card use by sensitive applications
• lists threats
• PP requirements:
  • 42 TOE security functional requirements
  • 24 TOE security assurance requirements
  • IT environment security requirements
  • with rationale for selection

Assurance
• “degree of confidence that the security controls operate correctly and protect the system as intended”
• applies to:
  • product security requirements, security policy, product design, implementation, operation
  • various approaches analyzing, checking, testing various aspects

Evaluation Assurance Levels
• EAL 1 - functionally tested
• EAL 2: structurally tested
• EAL 3: methodically tested and checked
• EAL 4: methodically designed, tested, and reviewed
• EAL 5: semiformally designed and tested
• EAL 6: semiformally verified design and tested
• EAL 7: formally verified design and tested

Evaluation
• ensure security features are correct and effective
• performed during/after TOE development
• higher levels need greater rigor and cost
• input: security target, evidence, actual TOE
• result: confirm security target satisfied for TOE
• process relates security target to some of TOE:
  • high-level design, low-level design, functional spec, source code, object code, hardware realization
  • higher levels need semiformal/formal models

Evaluation Parties & Phases
• evaluation parties:
  • sponsor - customer or vendor
  • developer - provides evidence for evaluation
  • evaluator - confirms requirements satisfied)
  • certifier - agency monitoring evaluation process
• phases:
  • preparation, conduct of evaluation, conclusion
• government agency regulates, e.g. US CCEVS
• have peering agreements between countries
  • saving time/expense by sharing results
Buffer Overflow

• a very common attack mechanism
  • from 1988 Morris Worm to Code Red, Slammer, Sasser and many others
  • prevention techniques known
  • still of major concern due to
    • legacy of widely deployed buggy
    • continued careless programming techniques

Buffer Overflow Basics

• caused by programming error
  • allows more data to be stored than capacity available in a fixed sized buffer
    • buffer can be on stack, heap, global data
  • overwriting adjacent memory locations
    • corruption of program data
    • unexpected transfer of control
    • memory access violation
    • execution of code chosen by attacker

Buffer Overflow Attacks

• to exploit a buffer overflow an attacker
  • must identify a buffer overflow vulnerability in some program
  • understand how buffer is stored in memory and determine potential for corruption

A Little Programming Language History

• at machine level all data an array of bytes
  • interpretation depends on instructions used
• modern high-level languages have a strong notion of type and valid operations
  • not vulnerable to buffer overflows
  • does incur overhead, some limits on use
  • C and related languages have high-level control structures, but allow direct access to memory
    • hence are vulnerable to buffer overflow
    • have a large legacy of widely used, unsafe, and hence vulnerable code

Function Calls and Stack Frames

Stack Buffer Overflow

• occurs when buffer is located on stack
  • used by Morris Worm
  • “Smashing the Stack” paper popularized it
• have local variables below saved frame pointer and return address
  • hence overflow of a local buffer can potentially overwrite these key control items
• attacker overwrites return address with address of desired code
  • program, system library or loaded in buffer
Shellcode

- code supplied by attacker
  - often saved in buffer being overflowed
  - traditionally transferred control to a shell

- machine code
  - specific to processor and operating system
  - traditionally needed good assembly language skills to create
  - more recently have automated sites/tools

Shellcode Development

- illustrate with classic Intel Linux shellcode to run Bourne shell interpreter
- shellcode must
  - marshall argument for execve() and call it
  - include all code to invoke system function
  - be position-independent
  - not contain NULLs (C string terminator)

More Stack Overflow Variants

- target program can be:
  - a trusted system utility
  - network service daemon
  - commonly used library code, e.g. image
- shellcode functions
  - spawn shell
  - create listener to launch shell on connect
  - create reverse connection to attacker
  - flush firewall rules
  - break out of chroot environment

Buffer Overflow Defenses

- buffer overflows are widely exploited
- large amount of vulnerable code in use
  - despite cause and countermeasures known
- two broad defense approaches
  - compile-time - harden new programs
  - run-time - handle attacks on existing programs

Compile-Time Defenses: Programming Language

- use a modern high-level languages with strong typing
  - not vulnerable to buffer overflow
  - compiler enforces range checks and permissible operations on variables
- do have cost in resource use
  - and restrictions on access to hardware
  - so still need some code in C like languages!

Compile-Time Defenses: Safe Coding Techniques

- if using potentially unsafe languages eg C
- programmer must explicitly write safe code
  - by design with new code
  - after code review of existing code, cf OpenBSD
- buffer overflow safety a subset of general safe coding techniques (Ch 12)
  - allow for graceful failure
  - checking have sufficient space in any buffer
Compile-Time Defenses: Language Extension, Safe Libraries

- have proposals for safety extensions to C
  - performance penalties
- must compile programs with special compiler
- have several safer standard library variants
  - new functions, e.g. strlcpy()
  - safer re-implementation of standard functions as a dynamic library, e.g. Libsafe

Compile-Time Defenses: Stack Protection

- add function entry and exit code to check stack for signs of corruption
- use random canary
  - e.g. Stackguard, Win /GS
  - check for overwrite between local variables and saved frame pointer and return address
  - abort program if change found
  - issues: recompilation, debugger support
- or save/check safe copy of return address
  - e.g. Stackshield, RAD

Run-Time Defenses: Non Executable Address Space

- use virtual memory support to make some regions of memory non-executable
  - e.g. stack, heap, global data
  - need h/w support in MMU
- long existed on SPARC / Solaris systems
- recent on x86 Linux/Unix/Windows systems
- issues: support for executable stack code
  - need special provisions

Run-Time Defenses: Address Space Randomization

- manipulate location of key data structures
  - stack, heap, global data
  - using random shift for each process
  - have large address range on modern systems means wasting some has negligible impact
- also randomize location of heap buffers
- and location of standard library functions

Run-Time Defenses: Guard Pages

- place guard pages between critical regions of memory
  - flagged in MMU as illegal addresses
  - any access aborts process
- can even place between stack frames and heap buffers
  - at execution time and space cost

Other Overflow Attacks

- have a range of other attack variants
  - stack overflow variants
  - heap overflow
  - global data overflow
  - format string overflow
  - integer overflow
- more likely to be discovered in future
- some cannot be prevented except by coding to prevent originally
Replacement Stack Frame

- stack overflow variant just rewrites buffer and saved frame pointer
- so return occurs but to dummy frame
- return of calling function controlled by attacker
- used when have limited buffer overflow
- e.g. off by one
- limitations
  - must know exact address of buffer
  - calling function executes with dummy frame

Return to System Call

- stack overflow variant replaces return address
  with standard library function
- response to non-executable stack defences
- attacker constructs suitable parameters on stack above return address
- function returns and library function executes
  - e.g. `system("shell commands")`
- attacker may need exact buffer address
- can even chain two library calls

Heap Overflow

- also attack buffer located in heap
- typically located above program code
- memory requested by programs to use in dynamic data structures, e.g. linked lists
- no return address
  - hence no easy transfer of control
  - may have function pointers can exploit
  - or manipulate management data structures
- defenses: non executable or random heap

Global Data Overflow

- can attack buffer located in global data
- may be located above program code
- if has function pointer and vulnerable buffer
- or adjacent process management tables
- aim to overwrite function pointer later called
- defenses: non executable or random global data region, move function pointers, guard pages

Questions