FORMALAPPROACHES TO SECURITY

Catherine Meadows
Code 5543
Center for High Assurance Computer Systems
US Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil
http://chacs.nrl.navy.mil

WHAT DO WE MEAN BY A FORMAL APPROACH TO SECURITY?

- **Security**
  - **can be broadly defined as correct behavior in face of an intelligent adversary or adversaries**
- **For a formal approach to security, we need**
  - **A well-defined specification of the system**
  - **An adversary model**
    - **Who are the adversaries?**
    - **What are their goals?**
    - **What are their capabilities?**
  - **A specification of correct behavior**
  - **An effective procedure for determining that the system behaves correctly in the face of the adversaries**
    - **"Effective" = can tell whether or not you've succeeded**

PLAN OF THESE LECTURES

- Begin with historical overview
  - Early work on time-sharing systems in 60's and 70's
  - DoD (Orange Book) security in 80's
  - Demise of Orange Book and what followed
- Look at some specific areas
  - Security Criteria
  - Access control
    - Including trust management
  - Information flow and non-interference
  - Proof-carrying code
    - Covered in more detail later in this course
  - Crypto protocol verification
    - Covered in more detail later in this course

EARLY WORK IN SECURITY

- For earliest computers, security not much of an issue
  - Few people would have access to a single computer
  - Those who did trusted each other
- Interest in security first motivated by introduction of time sharing systems
  - Multiple users with virtual access to a computer
    - Most without physical access
  - Examples:
    - Computer used by a students and faculty at a university
    - Computer used to manage accounts at a company
  - Computers not networked, or only in local, fixed networks
  - Users given access to only part of the system
  - Perceived threat
    - Legitimate user or software that manipulates system to get accesses to which it is not entitled

ACCESS MATRIX MODELS (LAMPSON 71)

- Three principal components
  - Subjects
  - Objects
    - Objects may also be subjects
  - Rights subjects may have to objects
    - Read, write, execute, etc.
- Components arranged in a matrix
  - One row per subject
  - One column per object
  - Right r appears in i'th row and j'th column if subject i has right r to object j
  - A configuration of the matrix is called a protection state
- Model also includes rules for moving from one protection state to another
  - Example: If subject i has "own" right to object j, then it can grant read access to j to any other subject

HARRISON, RUZZO, AND ULLMAN (1976)

- Undecidability of the safety problem
- HRO describe an access matrix model with six rules s.t. the following problem is undecidable:
  - Given an initial configuration, is it possible to generate a matrix with an arbitrary right in an arbitrary position?
- Has effect on general usefulness of model
  - How do you prevent subjects from having rights you don't want them to have?
- Will revisit the topic of decidable access control models later

LATTICE MODEL

- Inspired by DoD multilevel security
- Can think of security labels (Secret Secret/Red Secret/Blue Secret/Red&Blue, etc.) forming a lattice
  - Each label is a pair consisting of a security level (e.g. secret, top secret, etc.) and a compartment set
  - (a,b) ≥ (c,d) if a ≥ c and b ≥ d
  - Lub[(w,x),(y,z)] = (max(w,y), x ∪ z)
- Subjects can be granted access to data with appropriate security labels according to system policy
- Can represent lattice model within the access matrix model
  - More about lattics models and decidability later

INFORMATION FLOW MODELS (Fenton74, Denning75)

- Takes advantage of structure of lattice model
- Tracks information, instead of permissions
- Says that all information transfers must obey the partial order enforced by the lattice
  - Information can flow from variable x to variable y if and only if $l(y) \geq l(x)$
  - Example: f(x) := 1 if y = 0, f(x) := 0 otherwise
  - If x labeled secret, and y top secret, violates information flow
- Information flow has a long history, and is still going strong today
  - Much more on this later

SECURITY ARCHITECTURES

- How do you enforce these policies?
- We assume all subjects, even when executed by trusted users, are untrusted
  - Subjects are software that may have been supplied by untrusted third party
  - Not practical to verify all subjects secure, even when provenance is known
- Instead, have a reference monitor (Anderson72) which mediates all accesses, and only grants permission to those that satisfy the policy
- More generally, reference monitor part of a security kernel (Anderson72, Schell73) that contains all security-relevant parts of the system

IS THIS ENOUGH?
TWO INTRACTIBLE PROBLEMS IN COMPUTER SECURITY

- **Lampson: "A Note on the Confinement Problem"**
- **Thompson: "Reflections on Trusting Trust"**
- **Issues that will come up again and again**

A NOTE ON THE CONFINEMENT PROBLEM (LAMPSON 73)

Defined types of channels by which a rogue program could leak information to unauthorized users/programs

- **Legitimate channels**
  - **Example: rogue billing program prints sensitive information on a bill**
- **Storage channels**
  - **Storage that can be written to by the rogue program and read by unauthorized program**
  - **Example: using read/write locks to transmit a shared Boolean variable**
- **Covert channels**
  - **Those not intended for information transfer at all**
  - **Example: a service's effect on the system load**

Reflections on Trusting Trust (Thompson, 84)

- Ken Thompson's Turing Award Lecture
  - Describes a C compiler that, when it compiles itself, automatically inserts a Trojan Horse into the compiled code
  - Security flaw can only be detected by looking at object code, source code will show no trace of problem
- Moral: Can't be absolutely sure system is secure unless you've looked it at every level of detail

ONWARD INTO THE 80'S

- Biggest threat to computer security seen to be to sensitive government information
- Move to put documents on government computer systems meant that it could potentially be much easier to compromise security of classified information
- In late 70's, it was decided to establish a formal security and accreditation process:
  - Publish a set of criteria for security systems (Orange Book)
  - Set up an evaluation center (National Computer Security Center (NCSC), at NSA)
  - NCSC also funded research in computer security
    - Emphasis on research geared to Orange Book requirements
    - Most security research in US in the 80's funded by NCSC, so this had enormous impact
- Process was to be used to evaluate commercial as well as military systems
  - Intended to be voluntary

BELL-LAPADULA MODEL (73-75)

- Motivated by desire to have formally verifiable model that could be implemented using technology of the day
- Not mandated by Orange Book, but influenced it heavily
- Based on access matrix model
- Enforces multilevel security and discretionay policy

BLP MODEL

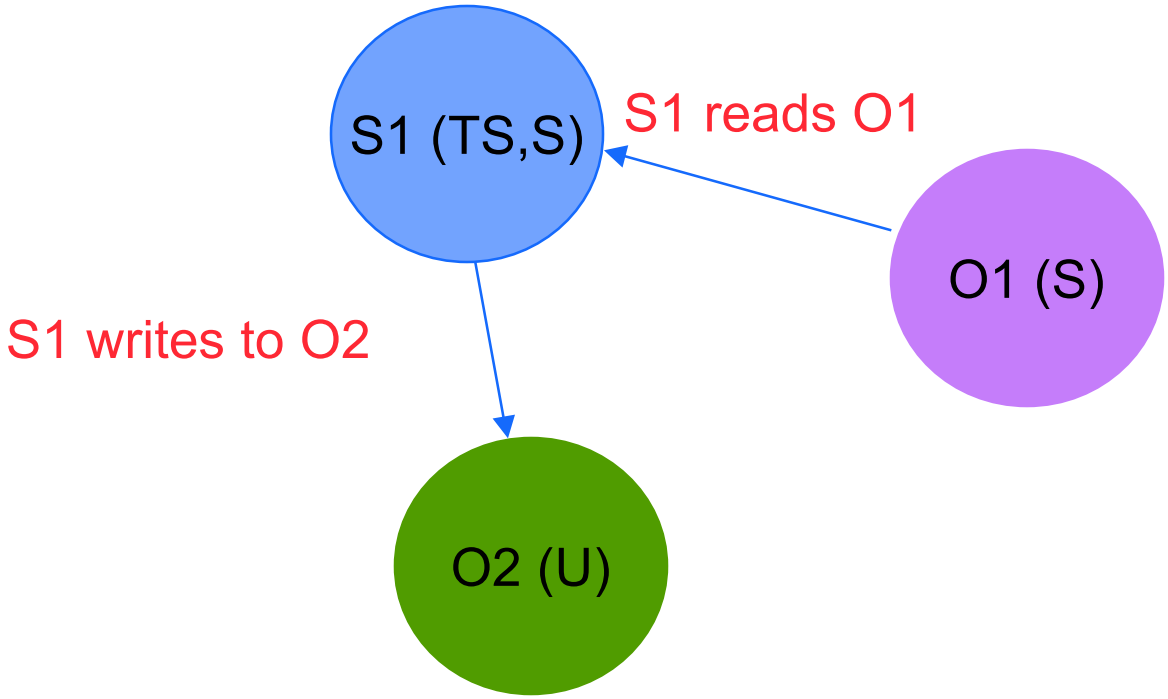
- Each subject has a clearance, each object has a classification
- Each subject has a current security level, which can't exceed its clearance
- Four modes of access
  - Read-only
  - Append
  - Execute
  - Read/write
- Control attribute allows subject who created object to pass on rights to other subjects
  - Control attribute cannot be passed on
- Creation in two parts
  - Addition of new inactive object to set of existing objects
  - Activation of inactive object
- Tranquility principal
  - No operation may change the security level of an active object
  - Note: this requirement not a part of military policy
  - Also not included in some versions of BLP

BLP DEFINITION OF SECURITY

- **Simple Security Property**
  - **If subject S has read access to object O, then the clearance of the subject must dominate the classification of)**
- *** Property**
  - **No subject S has append access to object O, unless the security level of O dominates the security level of S**
  - **No subject S has read-write access to object O, unless the security level of S is equal to the security level of O**
  - **No subject S has read access to object O, unless the security level of S dominates the security level of O**
- **Basic Security System**
  - **If system starts out in a secure state, all subsequent states secure**
  - **Secure state: one in which simple security and * property satisfied**

WHY THE * PROPERTY?

To prevent this:



CONCEPTS USED IN THE ORANGE BOOK

- Mandatory Access Control
  - Access control corresponding to DoD policy for classified data
  - Enforces a version of simple security and the * property
- Discretionary Access Control (DAC)
  - Allows users to define rights to objects (e.g. data)
  - Access rights at discretion of users
- Covert channels
  - Covert storage channels
    - Lampson's storage channels
  - Covert timing channels
    - Channels that use timing of events in system to signal information
    - Roughly corresponds to Lampson's covert channels
- Trusted Computing Base (TCB)
  - Part of system that must be trusted for system to be secure
  - Includes security kernel
  - Also includes parts of system "trusted" to violate security policy
    - E.g., to be exempt from * property

ORANGE BOOK CRITERIA CLASSES

D: no security

C1: Discretionary Security Protection

- TCB provides controls for enforcing access limitations

C2: Controlled Access Protection

- login procedures for users, auditing of security-relevant events, resource isolation

B1: Labeled Security Protection

- Informal statement of security policy model
- Data labeling
- Mandatory access control over named subjects and objects

B2: Structured Protection

- Formal Security Policy Model for TCB
- Covert channel analysis

B3: Security Domains

- TCB must mediate all accesses between subjects and objects
- TCB must be small enough to be subjected to analysis and tests

A1: Verified Design

- Formal Top Level Specification (FTLS) of TCB
- Formal verification that FTLS satisfies security policy model
- Informal demonstration that TCB satisfies FTLS

Some Problems

- How do you handle downgrading?
  - Not addressed in BLP, or in Orange Book
- How important are covert channels, really?
  - Removing them hurts system performance
    - So, only remove the big ones
    - Monitor the smaller ones
      - What does "monitor" mean?
  - Gave arbitrary criteria for acceptable bandwidth of channels
- How do you deal with networks?
  - Orange Book designed with standalone systems in mind
- How do you deal with multilevel applications?
  - Is a field in a database a single object?
- How do perform evaluations in a timely manner?
  - This was the biggest problem

REASONING ABOUT COMPUTER SECURITY (McLean, 87)

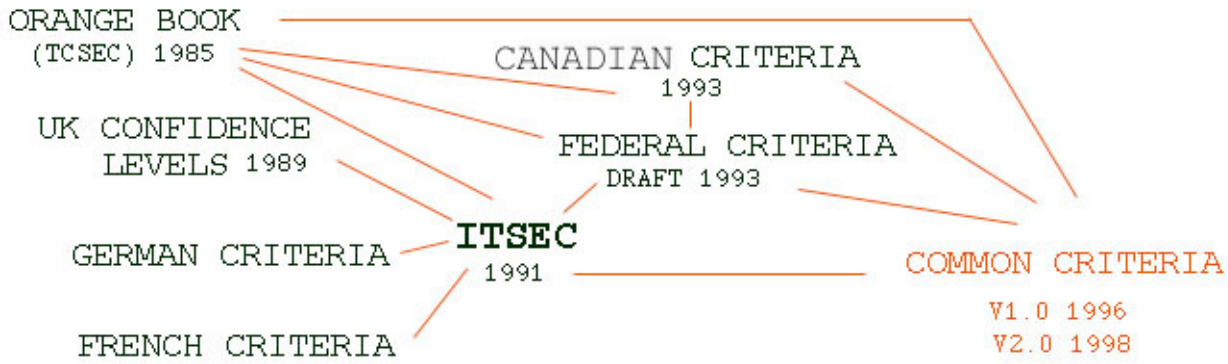
- Tackled the downgrading problem
- Posited a system, system Z, that downgraded all objects as soon as they became active
  - Violates tranquility principle, but this only present in some versions of BLP
- System Z trivially satisfied BLP Basic Security Theorem, but was obviously insecure
- So, what does BLP version of security give you?
  - If you include tranquility principle, you get a system at variance with DoD policy
  - If you leave tranquility principle out, you allow System Z

WHAT HAPPENED TO THE ORANGE BOOK?

- Overtaken by events
  - Difficult to apply to networked systems
    - Orange Book assumed system with a fixed boundary
    - Didn't address how different components of a system authenticate themselves to each other
  - Difficult (although not quite as hard) to apply to applications
    - Can you treat a field in a database as a BLP object and still get a high level of assurance?
  - Linking functionality and assurance hurt flexibility
    - What if you wanted A1 assurance for MAC and only B1 for DAC?
- Common Criteria developed to meet these deficiencies
- Orange Book continues to be used, but mainly to evaluate systems at C2 and B1 level

Common Criteria

- So-called because it incorporates several different sets of criteria
- Instead of meeting set of predefined criteria, one creates a security profile
- Functionality and assurance kept separate
- CC evaluations managed by international organization



STRUCTURE OF COMMON CRITERIA

- **11 Functional classes**
  - **Describe different classes of security functions system might need**
  - **Examples: audit, crypto support, user data protection, etc.**
- **8 Assurance classes**
  - **Examples: config. management, development, tests, etc.**
- **7 Evaluation assurance levels**
  - **Highest level: EAL7**
    - **Formally verified and tested**

EAL7 is applicable to the development of security TOEs for application in extremely high risk situations and/or where the high value of the assets justifies the higher costs. Practical application of EAL7 is currently limited to TOEs with tightly focused security functionality that is amenable to extensive formal analysis.

For an EAL7 evaluation, the formal model is supplementary by a formal presentation of the functional specification and high-level design, showing correspondence. Evidence of developer "white-box" testing and complete independent confirmation of developer test results are required. Complexity of the design must be minimized.

HOW COMMON CRITERIA USED

- Write protection profile (PP)
  - Describes security requirements for class of targets of evaluation
  - Should contain both functional and assurance requirements
  - Can use predefined assurance levels, or create your own
    - Most common: use predefined levels, but with some exceptions
  - Orange Book can be considered a set of protection profiles
- Write security target (ST)
  - Describes security-relevant system functionality
- Evaluate products with respect to PP and ST
  - Either ones you've written, or ones that already exist
- PP's and ST's must be evaluated as well
- Evaluation performed by accredited laboratories

Noninterference (Goguen and Meseguer '81)

- Used to provide a general notion of information flow
- Based on state machine model consisting of
  - Set U of "users"
  - Set S of "states"
  - Set C of "commands"
  - Set O of "outputs"
  - Output function out: S x U --> O
    - Tells what output a given user can see in a given state
  - State transition function do: S x U x C -> S
    - User u executes command c on state s, moving to new state s'
  - Initial state s_0 of S

Definition of Noninterference

- Let G be a subset of the set U of users
- Let A be a subset of the set C of actions
- Let w in (UxC)* be a sequence of pairs of users and actions
  - $p_G(w)$ = sequence w' obtained by removing pairs (u,c) $\in$ W with u $\in$ G
  - $p_A(w)$ = sequence w' obtained by removing pairs (u,c) $\in$ W with c $\in$ A
    $p_{G,A}(w)$ = sequence w' obtained by removing pairs (u,c) $\in$ W with u $\in$ G and c $\in$ A
  - If u is a user, $[[w]]_u$ is the output to u after w is performed
- We say that Z does not interfere with another set G' of users, if, for all users in G' and w in (U x C)*:

$$[[w]]_u = [[p_Z(w)]]_u$$

- In other words, the output visible to a user should be the same whether or not "invisible" users and/or actions are present
  - In multi-level formulation, Z = set of "high" users, G' = set of "low" users
- In subsequent paper, Goguen and Meseguer proved an "unwinding theorem" that characterized the states in a noninterfering system

What First Version of Noninterference Did and Did Not Address

- Didn't address composition of systems
- Didn't address nondeterministic systems
- Did address dynamic security labels
  - Extended notion of noninterference to "conditional" noninterference

FIRST ATTEMPT (ALMOST*) TO APPLY NONINTERFERENCE TO AN ACTUAL SYSTEM

- Secure ADA Target (SAT)
- Operating system designed to meet requirements for A1 level
- Haigh and Young (S&P, 1986) reformulated in terms of noninterference
- SAT security policy actually three policies
  - MLS: simple security + * property
  - Type enforcement
    - Restricts subject's access to objects based on role subject is playing
    - Type of object can change: idea similar to downgrading
    - Path by which object moved through system called the assured pipeline
  - Discretionary access control
- Used noninterference to model first two
- Needed conditional noninterference to model type enforcement
  - Only look for covert channels outside the assured pipeline

* Goguen and Meseguer performed an unpublished analysis of NRL's Military Message System Model

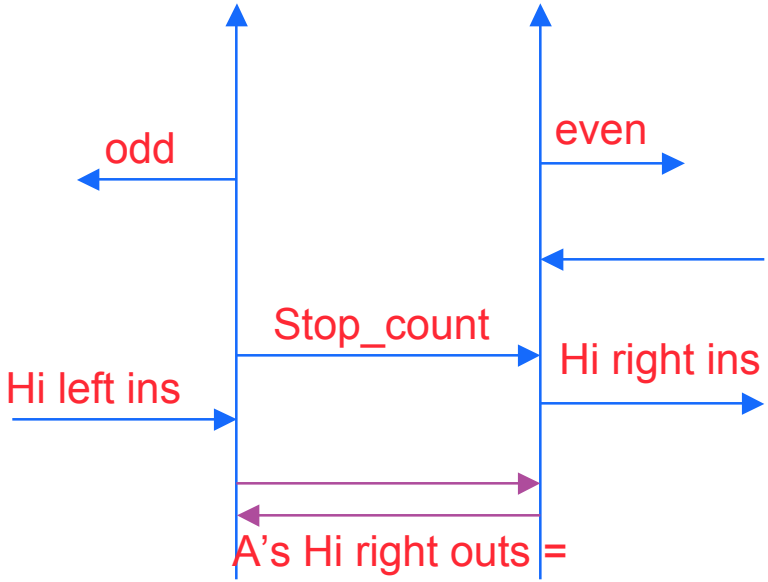
COMPOSITION OF NONINTERFERENCE (McCullough, 87)

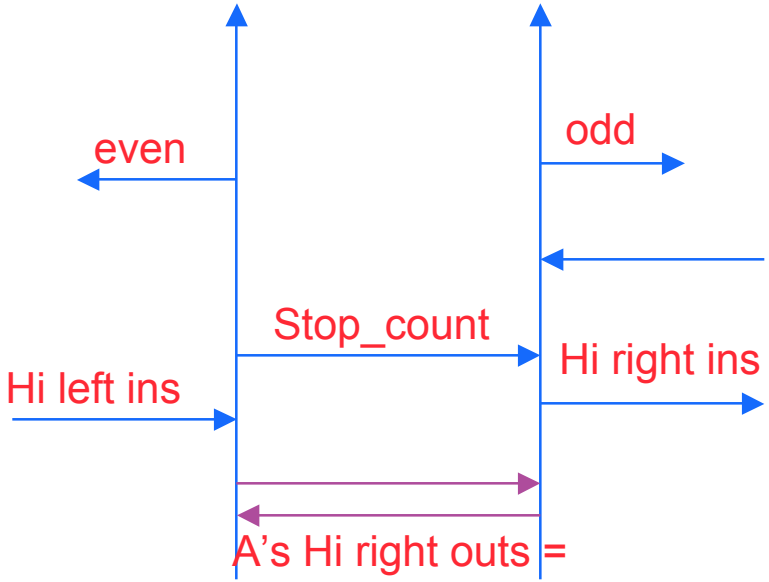
- Defined system in terms of what sequences of inputs and outputs possible
  - Allowed one to define nondeterministic systems more naturally
- Defined what seemed to be a natural extension of noninterference to nondeterministic systems
  - A system is satisfies generalized noninterference if
    - For all traces σ_1 and all sequences σ_2 formed from σ_1 by adding or deleting high-level inputs
  - There is a trace σ_3 such that if
    - We write σ_1 as σ ^ σ_1 and σ_2 as σ ^ σ_2, then there is a trace σ ^ σ_3 where σ_2 and σ_3 the same if we remove high-level outputs
- Intuitively, always a trace with same low-level outputs no matter what we do to high-level inputs

A COUNTEREXAMPLE TO COMPOSABILITY

- System A
  - Each trace consists of
    - A sequence of high_level inputs or outputs
      - followed by low-level output stop_count
    - Followed by low-level output odd if number of high-level before stop_count odd, by low-level output even otherwise
  - High level outputs and stop_count leave via right channel
  - High level inputs enter via left or right channel
  - "Odd" and "even" outputs leave via left channel
- System B, just like A, except
  - High-level outputs leave via left channel
  - "Even" and "odd" outputs leave via right channel
  - Stop_count input to its left channel
- Both Systems A and B satsify generalized noninterference
  - Intuitively, "stop_count" doesn't tell you parity of high-level inputs
    - in.out.stop_count.even
    - in.in.stop_count.even
  - Are both OK
- Trojan Horse sending inputs to A or B cannot use it to signal to low

COMPOSING A AND B





In both cases where A and B differ after stop_count,
must be at least one high-level input from outside

A Composable Property: Restrictiveness

- A system is satisfies restrictiveness if
  - For all traces β_1 and all sequences β_2 formed from β_1 by adding or deleting high-level inputs
- There is a trace β_3 such that if
  - We write β_1 as α ^ β_1 ^ γ_1 and β_2 as α ^ β_2 ^ γ_2 , where
    - α longest common initial segment of β_1 and β_2
    - β_1 longest sequence of inputs following α in β_1
    - β_2 longest sequence of inputs following α in β_2
  - Then α β_3 = α ^ β_2 ^ γ_3 where γ_3 differs from γ_2 only in the high-level outputs
- Problem with restrictivenness - not very intuitive

SELECTIVE INTERLEAVING FUNCTIONS (McLean '94)

- Satisfies different noninterference properties in terms of closure behavior of interleaving functions
- Example: Separability: satisfied if, for any trace t1, can obtain a trace t2 by deleting all events of level l and replacing with events of a single-level trace of level l
- Consider F : S x S -> S such that F(t1,t2) = t =>
  - Highin(t) = highin(t1), Lowin(t) = lowin(t2)
  - Highout(t) = highout(t1), Lowout(t) = lowout(t2)
- A system satisfies separability if it is closed under F
- Can characterize other noninterference properties in terms of different interleaving functions
- Goes on to characterize composition in terms of six constructs
  - External: product, cascade, feedback
  - Internal: Union, intersection, and difference of allowable events
- Characterizes closure under composition in terms of closure under composition of appropriate selective interleaving functions

HIATUS IN NONINTERFERENCE RESEARCH FOR SEVERAL YEARS

- Pretty much had said everything you could say about nondeterministic noninterference
- Less interest in multilevel security in general
  - Research focused more on security of networks, commercial security, etc.
- Hard to design real systems that satisfied noninterference
- Recently, have seen resurgence of interest in noninterference
  - Applicability to other areas (e.g. crypto protocol analysis)
  - Emergence of formal systems such as process algebras
    - Make it easier to specify and reason about properties such as non interference

PROBABILISTIC NONINTERFERENCE

- **Motivation: consider the following system satsifying separability**
  - **Inputs hi_in**
  - **Outputs lo_out**
  - **Any sequence of hi_in and lo_out possible**
  - **Suppose that probability of lo_out occurring is .8 if no hi_in occurred previously, .001 if a hi_in occurred previously**
  - **System clearly leaks information about high inputs**
- **What's needed: a notion of noninterference that takes probability into account**

PROBABILISTIC BISIMULATIONS

- First used for noninterference by Sabelfeld and Sands (CSFW, 2000)
- Let M be a Markov chain in which all state variables have a security label, high or low
- An equivalence relation on M is a probabilistic bisimulation on M if for all equivalence classes A and B, the probability of going from a state a in A to a state in B is independent of a
- Define a probabilistic bisimulation such that two states are in the same equivalence class if and only if agree on low variables
- Probabilistic Markov chain is noninterfering if any sequence of states always in the same equivalence class

- Allows you to talk about probabilistic systems, but what about quantifying interference?

APPROXIMATE NON-INTERFERENCE (Di Pierro, Hankin, Wiklicky, 2002)

- Start with a probabilistic transition system
- Given a process A and initial state d, define O(A) corresponding to the set of all pairs <c,p>
  - c result of finite computation of A starting in initial state
  - p probability of computing that result
- Given two processes A and B, define ||O(A) - O(B)|| to be the max of all |pA - pB| such that <c,pA> in O(A) and <c,pB> in O(B)
- Given two processes A and B, and a set of passive, memoryless observers S, and fixed scheduling priorities x and y, we say that A and B are ε-confined it

$$\sup_{s \in S} \| O(x:A \| y: s) - O(x:B \| y:s) \| < \varepsilon$$

- Also show how to compute most effective observers

ANOTHER APPROACH: MODELLING COVERT TIMING CHANNELS (Lowe, 2002)

- Covert channel analysis (as opposed to information flow analysis) works by
  - Constructing a mathematical model of principals exploiting leak in system to pass information
  - Computing capacity and/or bandwidth of channel they create
- Lowe's approach: model high and low participants in covert channel as discrete-time CSP processes
- Rate of information flow is the number of different values High can send to Low
- Corresponds well to notion information-theoretic notion of capacity

SOME OPEN PROBLEMS IN NONINTERFERENCE

- Application to realistic information flow problems
  - Original application, operating systems, probably too large
  - Other possible applications
    - Smart cards
    - One-way flow devices
    - Already seen application in crypto protocol analysis

- Application to related problems in information hiding
  - Steganography
  - Cryptography
    - Subliminal channels
    - Attacks on crypto hardware
      - Fault injection, power attacks, etc.

ISSUES IN ACCESS CONTROL

- **Decidable access control systems**
  - **Safety problem should be decidable**
- **Realistic access control systems**
  - **Should be able to express realistic security policies**
- **Network friendly access control systems**
  - **Access control should be enforcable in a modern networked environment**
- **User friendly access control**
  - **Systems should be easy to use correctly**

SOME DECIDABLE SYSTEMS

- Mono-conditional monotonic commands
  - No deletes or destroys permitted
  - Conditions may have only one operation
- Mon-operational command (NP-complete)
  - Condition can be arbitarily complex but actual command either
    - Add or delete single right from entry
    - Create or delete single row or column
- BLP MAC
- Take-grant model (linear)

TAKE-GRANT MODEL (Lipton and Snyder, 1977)

- System represented by a directed graph
- Subjects and objects are vertices
- Access rights edges
  - Two distinguished rights take, and grant
- Rules:
  - Take rule: If subject A has take right to B, and B has right a to C, then A can obtain right a to C
  - Grant rule: If subject A has grant right to B, and right a to C, then A can B right a to C
  - Create rule: Any subject A can create a node B and grant itself any rights it wants to B
  - Remove rule: A can delete any rights it has to B
- Theoretical interest, but did not lead to practical systems

Schematic Protection Model (Sandhu)

- Monotonic - revocation of rights not allowed
  - Revocation of rights can be ignored for purposes of safety analysis
- Finite set of types
- Each subject or object has a type that can't be altered
- Can-create relation cc(T1,T2)
  - A subject of type T1 can create an object of type T2 if an only if cc(T1,T2) holds
- Cc-graph: directed graph such that
  - Vertices are types
  - Edge from T1 to T2 if and only if cc(T1,T2) holds
- If cc-graph acyclic, safety is decidable
  - Also decidable for certain types of cycles of length 1
- Extended SPM allows multiple parents for a child
  - Has similar decidability properties to SPM
  - Equivalent to monotonic HRU

EXPRESSING REALISTIC POLICIES - ROLE-BASED ACCESS CONTROL

- Realized early on that MAC too restrictive for most policies and DAC not restrictive enough
- In most real-life systems, people play various roles
  - Rights are granted to them as part of role being played
  - Example
    - Doctor has right to see record of patient
    - Patient has right to see own record
- Early system to formalize this policy for a particular application
  - NRL military message system (MMS) 1984
- What's needed: a general model for expressing roles-based access policies

RBAC (Sandhu et al.)

- System consists of
  - Users
  - Permissions
  - Roles -- collections of permissions
  - Sessions
- Each user granted one or more roles
  - Can take on subset of roles during a session
- Other features of roles
  - Hierarchical relation
    - Possible to have inheritance of permissions from one role to another
    - E.g. a surgeon is a type of doctor
  - Constraints on use of roles
    - For example, user may have access to two roles, but not able to take on both roles at the same time (e.g. doctor and patient)

ACCESS CONTROL IN NETWORKS

- Earlier access control policies assumed a central authority or at worst a group of authorities enforcing access control policy
- In networking world access control more likely to be enforced by digital certificates
- An entity will receive a set of signed credentials from a principal making a request for access
  - Will make decision to grant or deny access based on these credentials
- How do we make sure that credentials express coherent policy?
- How do we make sure credentials express policy umambiguously?
- How do we decide when to believe credentials?
- This is the problem of trust management.

ISSUES TO BE ADDRESSED IN TRUST MANAGEMENT (Blaze et al.)

- **Authentication**
  - Need mechanisms for authenticating credentials
  - Need mechanisms for authenticating principals specified in credentials
- **Delegation**
  - If principals have the ability to grant privileges, should be able to delegate this to another principal
    - Should be able to do this in such a way that high-level entities can specify overall policy, but leave detail to lower-level entities
  - Necessary for scalablity
- **Expressibility and Extensibility**
  - Must be able to handle new conditions and restrictions on the fly
- **Local trust policy**
  - Different locations may trust different authorities
- **Unambiguity**
  - Although credentials may be distributed, the policy they specify should be unambiguous when they are combined

PGP WEB OF TRUST

- User generates pair (P,S) consisting of public key P and secret key S
  - Associates pair with unique ID
  - Stores keys (both public and private) in key records
- Key record contains as ID, public (or private) key, and timestamp of when key created
- Signed key record called a certificate
- A principal A can act as an introducer of a principal B by
  - Signing the record of B's public key
  - Specifying the degree of trust (optional) in the binding of B's identity to key in record
    - Unknown, untrusted, marginally trusted, trusted
- Problems
  - Not suitable for specifying access control policies
  - Doesn't scale well

X.509 Authentication Framework

- X.509 certificates signed records like PGP's
  - Contain a little more information than PGP certificates
    - Names of signature schemes used to create them
    - Time intervals in which valid
- Main difference: certificates can only be created by official certifying authority (CA)
  - If A wants to communicate securely with B, obtains certificate from directory server
  - If A and B both certified by common CA, server sends B's certificate to A, who checks CA's signature
  - If A and B not directly certified, service must create certification path to B
    - Find a common ancestor of A's and B's CA's
    - Traverse the tree from A's CA to the ancestor and back down to B's CA checking certifcates as you go
- Drawback: requires tree structure in place before you can begin

POLICYMAKER

- Certificates bind public keys to predicates describing actions they are trusted to sign for
  - Authorize a crypto key to sign for purchase orders up to $500
  - Authorize a key to sign one and only one vote in an election
- Decision to accept certificates based on one's trust in key that signed the certificate

POLICYMAKER QUERIES

- **Request to determine whether a sequence of public keys sufficient to perform a particular action according to local policy**
- **key_1,…,key_n REQUESTS ActionString**

POLICYMAKER ASSERTIONS

- Assertions confer authority on keys
- Format: Source ASSERTS AuthorityStruct WHERE Filter
  - Source = source of assertion
    - Local policy: assertion originates locally
    - Public key: assertion is a certificate signed by corresponding private key
  - AuthorityStruct specifies key or keys to which assertion applies
    - Could put some logical structure on keys
      - K1 or K2 , K1 and K2, etc.
  - Filter = predicate action strings must satisfy for assertion to hold
- Semantics of assertion
    - States that source trusts public keys in authority structure to be associated with action strings that defined the filter

HOW POLICYMAKER PROCESSES QUERIES

- System must have at least one local policy assertion in place before it can process queries
  - Local assertion acts as “trusted root”
- Consider the following directed graph D
  - Vertices labeled by keys or policy sources
  - Arcs labeled by filters
  - If v -> w an arc labeled by f, then there is a assertion whose source is the label of v, whose authority structure is w, and whose filter is f
- A query is compliant with the policy if there is a chain v1 -> … -> vt where vi is a local policy source and vt labeled by an authority structure that accepts the keys specified in the query

COMPLEXITY OF COMPLIANCE CHECKING PROBLEM

- In general, undecidable
- NP-Hard cases
  - Locally bounded proof of compliance
    - Different bounds on sizes of different parts of the proof
  - Globally bounded (NP-complete)
    - Single bound on size of entire proof
  - Monotonic
    - If have set of evidence that proves assertion, so will any superset of evidence
- Locally bounded, Monotonic, and Authentic is polynomial time
  - "Authentic" a rather technical condition that says that credentials don't impersonate each other

Delegation Logic (Li)

- Language for specifying and reasoning about trust management based on logic programming
  - Rules of the form H if F
  - H statement about something a principal does (says,delegates or speaks-for)
  - F (body statement) statement about something a principal structure does
  - Principal structure: principal names conjoined by logical operators
- Semantics based on model theoretic semantics of logic programming
- Monotonic and non-Monotonic versions
  - Restrictions on non-monotonic versions allow tractable complexity results
    - Classical negation can be used before simple predicates
    - Negation as failure can be used before body statements

SOME STANDARDS IN THE WORKS: SPKI

- Stands for Simple Public Key Infrastructure
- Uses certificates directly for authorization
- Gives standard format, some what more restrictive than Policymaker
  - However, allows for restricted form of nonmonoticity through validity dates
- Five fields
  - Issuer
  - Subject
  - Delegation (says whether subject able to pass on authority)
  - Authorization (what the subject is being authorized to do)
  - Validity dates

SOME STANDARDS IN THE WORKS: SDSI

- **Stands for Simple Distributed Security Infrastructure**
- **Used for specifying names in certificates**
  - **Compatible with SPKI certificates**
- **No global name space**
- **Provides means for linking local name spaces**
  - **E.g. Bob's alice (or ref: bob alice) refers to the principal Bob calls Alice**
  - **Bob exports binding issuing a signed certificate binding his local name Alice to that principal's key**
  - **Can have string of local names**
    - `GE's lighting-division's vp-of-marketing's secretary's assistant`

ACCESS CONTROL CONSTRUCTS SUPPORTED BY SDSI

Groups

- Either by listing members or algebraic expression involving other groups
  - `( Group: ( OR: faculty staff Bob-Smith ) )`

Roles

- Define public key for each role
- Assign role to principal by signing certificate with role public key

Delegation

- Gives delegee authority to sign certain types of certificates on behalf of delegator

MODEL-CHECKING SPKI/SDSI (Jha and Reps)

- Showed that SPKI/SDSI could be modeled as a pushdown system (PDS)
- PDS similar to pushdown automaton, w/0 input alphabet
- PDS = triple P = (P,□,□)
  - P = finite set of control locations
    - SPKI/SDSI keys and resources correspond to control locations
  - □ = finite stack alphabet
    - Alphabet is set of identifiers + delegation bits
  - □ □ (P x □) x (P x □*) = finite set of transition rules
    - Certificates are transition rules
- Problem of determining whether a subject is authorized to perform a particular action becomes problem in model-checking a PDS
- By using known results about model-checking PDS, found low-degree polynomial-time algorithms for solving authorization problems in SPKI/SDSI

HOW INTERVALS OF VALIDITY REPRESENTED IN PDSs

- Think of time intervals as a lattice
  - Meet of two intervals is theire intersection
  - Join of [n,m] and [n',m'] is [min(n,n'),max(m,m')[
- Label each transition corresponding to a certificate with that certificate's validity interval
- Label corresponding to a path is meet of all labels in that path
- Since algorithm works by constructing paths, straightforward to add testing for intervals of validity

FINAL ISSUE IN ACCESS CONTROL - USER FRIENDLINESS

- Not necessarily a formal methods issue, *but*
- Ff access control systems not understandable and usable, it doesn't matter how sound and expressive they are
- Example:
  - "Why Johnny Can't Encrypt" Whitten and Tygar, USENIX 2000
  - Case study of PGP
  - Found that graphical user interface did not clearly describe actual policy supported by PGP
  - When tested on actual users found that many
    - Couldn't figure out how to encrypt at all
    - Couldn't figure out correct key to encrypt with
    - Couldn't decrypt email
    - Couldn't publish their public key
    - Etc.,
- Need better way of communicating conceptual models to users

CONCEPT OF SAFE STAGING APPLIED TO SECURITY (Whitten and Tygar 2003)

- Progress through system in stages
- Initial stage: Combine restrictive but safe policy with intuitive strategy for protecting assets
- Don't let user move on to next stage until sure that they understand risks
- Applied approach to public key certification
  - Stage 1: start out with weak strategy for verifying identity
    - References to shared knowledge, writing style, etc.
    - Used to motivate Stage 2, in which certification of public keys introduced
- Question: Are there ways of designing formal access control systems that would support a staging process like this?

Proof-Carrying Code -- Outline

- Describe motivation of problem early work on mobile code
  - Problems in Java security
- Brief overview of research in proof-carrying code

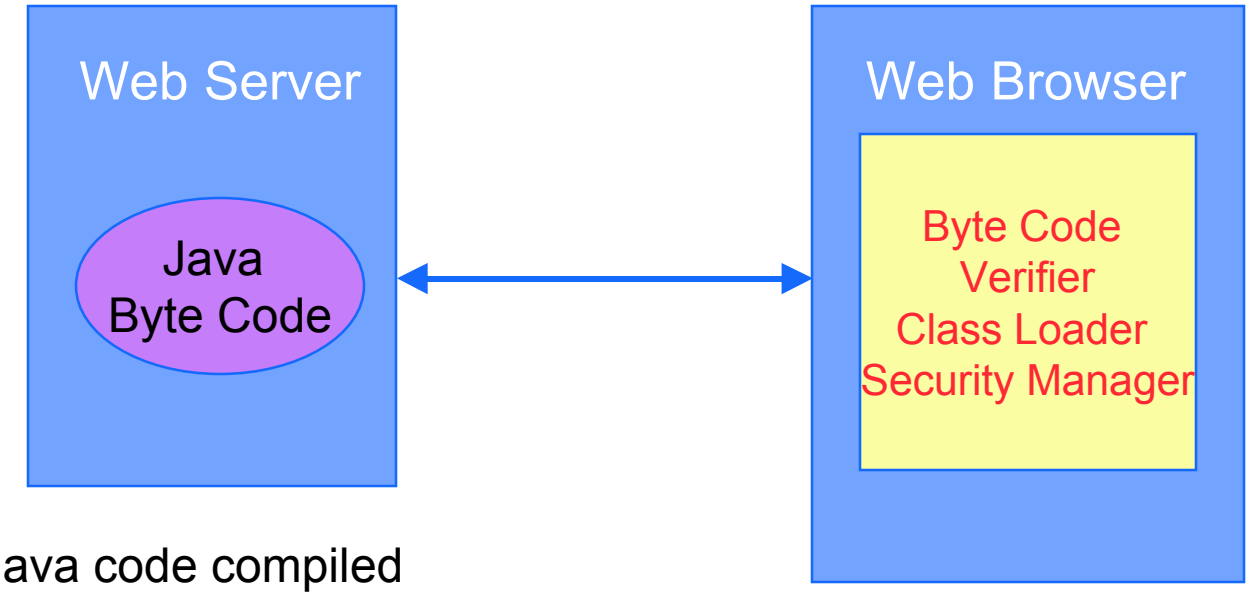
SECURITY ISSUES IN MOBILE CODE

- **Mobile code - code that that roams the Internet, executing at different nodes**
- **How do you trust mobile code?**
  - **May not be able to verify where it's from**
  - **May not be able to verify what it's doing or capable of doing**
    - **May be able to read, alter, delete files**

HOW JAVA ENFORCES SECURITY

- **Object-oriented**
  - **Data objects possess associated methods**
  - **Methods perform actions on data objects**
  - **Objects collected in classes**
    - **Methods defined for objects in a class**
  - **Classes arranged in hierarchy so that they inherit behavior from superclasses**
- **Java strongly typed**
  - **Program cannot access memory arbitrarily**
  - **Every piece of memory some Java object**
  - **Program cannot perform operation on an object unless it's valid for that object**

JAVA SECURITY ARCHITECTURE



Java code compiled to platform-independent Byte Code

1. Byte Code verifier checks type safety
2. Class loader installs code in separate name space
3. Security Manager checks all access requests at run-time

WHY TYPE SAFETY MATTERS (McGraw and Felten)

- Suppose a calendar-manager applet defines a class Alarm
  - Alarm defines operation turnOn setting first field to true
- Java run-time library defines another class called Applet
  - First field of Applet is fileAccessAllowed
  - Tells you whether or not applet allowed to access files on hard disk
- If program allowed to apply turnOn procedure to Applet object, would set first field to true
  - This would give the applet access to the hard disk
- Numerous examples of type flaws resulting in serious security breaches

AN EXAMPLE OF TYPE CONFUSION (Drew Dean)

- Java allows a program that uses a type T to use a type array of T
  - Array types not explicitly declared, but created on the fly when compiled
  - Java gives them name beginning with character [to avoid confusion with other types
- In Netscape Navigator 2.0□5, a Java byte code file could declare its own type name to one beginning with [
  - Attempting to load that class would cause an error, but name would be in internal table anyway
- Result: Java considered object an array, but had some other type

WHY ARE TYPE FLAWS SO COMMON?

- Two approaches to checking for type safety
  - Dynamic
    - Check at moment of performing method
    - Reliable but slow
  - Static
    - Check at compilation
    - Faster, but more complex, hence less reliable
- Guess which one was chosen?

ANOTHER APPROACH: PROOF-CARRYING CODE (Necula and Lee)

- Too much to ask receiver of mobile code to verify that it is correct
- But, it's often easier to check a proof than generate it oneself
- The writer of the code could generate a proof and send it with the code
  - Proof could be generated by the compiler and sent with the compiled code
- Receiver of code could then check that the proof is valid

ADVANTAGES OF PROOF-CARRYING CODE

- More reliable than static checking
  - It's static checking with validation from the receiver of the code
- Faster and potentially more flexible than run-time checking
- Doesn't require you to trust sender
- Difficult to tamper with
  - Tampering will result in invalid proof

CHALLENGES OF PROOF-CARRYING CODE

- Verifying that code is secure
  - Build certifying compiler to prove type safety as the code is compiled
  - Could also use theorem-prover, or even hand proofs
- Keeping proofs small
  - Proofs need to be sent with code
- Keeping proof checking inexpensive
- Increasing range of security policies supported while keeping the above three factors in mind
- Requires some hard, focused work on theorem proving
- Stick around next week to find out more!

APPLICATIONS OF FORMAL METHODS TO SECURITY IN THE LARGE

- Most of the technique we're described so far apply to relatively small examples
  - Formal methods expensive, so it's best to think small
- But, many security problems only apparent in the large
  - For example, many network attacks rely on exploiting multiple vulnerabilities in multiple systems
- Is there any way we can get some help from formal methods in this case?

USING MODEL-CHECKING FOR NETWORK VULNERABILITY ANALYSIS (Sheyner et al., 2002)

- Model network as finite state machine
- Model atomic attacks as state transitions
  - Sshd buffer overflow
  - ftp .rhosts
  - Remote login
  - Local buffer overflow
- Use model checker to generate attacks
- Some variants
  - Look for minimal sets of atomic attacks whose prevention would make goal unobtainable by the intruder
  - If have information about probabilities of attacks, can perform probabilistic reliability analysis by modeling attack graph as Markov Decision Process

References

D. Bell and L. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation, MTR-2997, March 1976. Available at http://www.math.chalmers.se/~dave/foundations/

M. Bishop and L. Snyder, "The Transfer of Information and Authority in a Protection System," Proceedings of the Seventh Symposium in Operating Systems Principles pp. 45-54 (Dec. 1979). Available at http://nob.cs.ucdavis.edu/~bishop/

M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. "The Role of Trust Management in Distributed Systems Security." Chapter in Secure Internet Programming: Security Issues for Mobile and Distributed Objects, (Vitek and Jensen, eds.) Springer-Verlag, 1999. Available at http://www.crypto.com/papers/

Dorothy Denning. A Lattice Model of Secure Information Flow, Comm. of the ACM, Vol. 19, No. 5, May 1976. Available at http://www.cs.georgetown.edu/~denning/publications.html

Department of Defense Trusted Computer Systems Evaluation Criteria, DoD 5200.28-STD December 1985. Available at http://www.dynamoo.com/orange/

Alessandra Di Pierro, Chris Hankin,and Herbert Wiklicky. Approximate Non-Interference, 15th IEEE Computer Security Foundations Workshop, 2002.

J. S. Fenton. An Abstract Computer Model Demonstrating Directional Information Flow, U. of Cambridge, 1974.

References (Cont.)

Gavin Lowe, Quantifying Information Flow, 15th IEEE Computer Security Foundations Workshop, 2002.

J. Goguen and J. Meseguer. Security Policies and Security Models, Proceedings 1982 Symposium on Security and Privacy, IEEE Computer Society Press, May 1981.

J. T. Haigh and W. D. Young. Extending the Non-Interference Version of MLS for SAT, 1986 IEEE Symposium in Security and Privacy. 1986.

M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. Communications of the ACM, 19(8):461-471, 1976.

S. Jha and T. Reps, Analysis of SPKI/SDSI Certificates Using Model Checking, Computer Security Foundations Workshop (CSFW), June 2002. Available at http://www.cs.wisc.edu/~jha/security.html

B. Lampson. Protection. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1 (Jan. 1974), pp 18-24. Available at http://research.microsoft.com/lampson/Publications.html

Butler Lampson. A note on the confinement problem. Comm. ACM 16, 10 (Oct. 1973), pp 613-615. Available at http://research.microsoft.com/lampson/Publications.html

Carl Landwehr. "Formal Models for Computer Security," ACM Computing Surveys, Vol. 13, Number 3 (September, 1981).

References (cont.)

Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. ACM Transactions on Information and System Security (TISSEC), volume 6, number 1, pp. 128-171, February 2003. Available at http://crypto.stanford.edu/~ninghui/

Richard J. Lipton, Lawrence Snyder: A Linear Time Algorithm for Deciding Subject Security. JACM 24(3): 455-464 (1977)

Daryl McCullough. A hookup theorem for multilevel security. IEEE Transactions on Software Engineering, 16(6):563--568, June 1990.

Gary Mcgraw and Edward Felten, Java Security. Wiley, 1997.

John McLean. Reasoning about Security Models. Proc. 1987 IEEE Symposium on Research in Security and Privacy, IEEE Press, 1987.

John McLean.. "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions," Proceedings of 1994 IEEE Symposium on Research in Security and Privacy, IEEE Press, 1994.

George C. Necula and Peter Lee. "Research on Proof-Carrying Code for Untrusted-Code Security". In Proceedings of the 1997 IEEE Symposium on Security and Privacy, Oakland, 1997. Available at http://www-2.cs.cmu.edu/~necula/papers.html

Andrei Sabelfeld and David Sands. Probabilistic Noninterference for Multi-threaded Programs, 13th IEEE Computer Security Foundations Workshop, 2000.

References (cont.)

Ravi Sandhu, The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes , Journal of the ACM, Volume 35, Number 2, April 1988. Available at http://www.list.gmu.edu/journal_papers.htm

Ravi Sandhu, Expressive Power of the Schematic Protection Model, Journal of Computer Security, Volume 1, Number 1, 1992. Available at http://www.list.gmu.edu/journal_papers.htm

Ravi Sandhu, Edward Coyne, Hal Feinstein and Charles Youman, Role-Based Access Control Models, IEEE Computer, Volume 29, Number 2, February 1996. Available at http://www.list.gmu.edu/journal_papers.htm

O. Sheyner, J. Haines, S. Jha, R. Lippman, and J.M. Wing, Automated Generation and Analysis of Attack Graphs, IEEE Symposium on Security and Privacy, April 2002. Available at http://www.cs.wisc.edu/~jha/security.html

Ken Thompson, Reflections on Trusting Trust, Communication of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763. Available at http://www.acm.org/classics/sep95/

Alma Whitten and J.D. Tygar, Why Johnny Can't Encrypt. Proceedings of the 9th USENIX Security Symposium, 1999. Available at http://www.gaudior.net/alma/

Alma Whitten and J. D. Tygar, Safe Staging for Computer Security. Chi 2003, 2003. Available at http://www.gaudior.net/alma/