# Course Outline

<table>
<thead>
<tr>
<th>Course</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Introduction: Role of security, Types of security, Definitions.</td>
</tr>
<tr>
<td>2.</td>
<td>Classification Schemes, Access Control.</td>
</tr>
<tr>
<td>5.</td>
<td>Formal policy models.</td>
</tr>
<tr>
<td>8.</td>
<td>Forensics</td>
</tr>
<tr>
<td></td>
<td><strong>Midterm</strong></td>
</tr>
<tr>
<td>9.</td>
<td>TCB and security kernel construction, Verification, Certification issues.</td>
</tr>
<tr>
<td>10.</td>
<td>System Verification.</td>
</tr>
<tr>
<td></td>
<td>Distributed authentication issues.</td>
</tr>
<tr>
<td></td>
<td>Routing, flooding, spamming. Firewalls.</td>
</tr>
<tr>
<td>13.</td>
<td>Malicious Code: Viruses, Worms, etc.</td>
</tr>
<tr>
<td>14.</td>
<td>Intrusion Detection and Response</td>
</tr>
<tr>
<td>15.</td>
<td>Vulnerability Analysis.</td>
</tr>
<tr>
<td>16.</td>
<td>Buffer overflow, secure programming</td>
</tr>
</tbody>
</table>

**Final Exam**

12/10/2010
Access Control Matrix
(Harrison/Ruzzo/Ullman)

- State: Status of the system
  - Protection state: subset that deals with protection
- Access Control Matrix
  - Describes protection state
- Formally:
  - Objects $O$
  - Subjects $S$
  - Matrix $A \subseteq S \times O$
- Tuple $(S, O, A)$ defines protection states of system

Protection State Transitions

- State $X_i = (S_i, O_i, A_i)$
- Transitions $\tau_i$
  - Single transition $X_i \xrightarrow{\tau_{i+1}} X_{i+1}$
  - Series of transitions $X \xrightarrow{\ast} Y$
- Access control matrix may change
  - Change command $c$ associated with transition
  - $X_i \xrightarrow{c_{i+1} (p_{i+1}, \ldots, p_{i+1})} X_{i+1}$
- Change command $c$ associated with transition
Primitive Commands

- **Create Object o**
  - Adds o to objects with no access
  - \( S' = S, O' = O \cup \{o\} \), \( (\forall x \in S')[a'[x,o] = \emptyset] \),
  - \( (\forall x \in S')(\forall y \in O)[a'[x,y] = a[x,y]] \)

- **Create Subject s**
  - Adds s to objects, subjects, sets relevant access control to \( \emptyset \)

- **Enter r into a[s,o]**
- **Delete r from a[s,o]**
- **Destroy subject s, destroy object o**

Formally:

- **Given**
  - initial state \( X_0 = (S_0, O_0, A_0) \)
  - Set of primitive commands \( c \)
- **Can we reach a state \( X_n \) where \( \exists s,o \) such that \( A_n[s,o] \) includes a right \( r \) not in \( A_0[s,o] \)?**
  - If so, the system is not safe
Decidability Result
*(Harrison, Ruzzo, Ullman)*

- Given a system where each command consists of a single *primitive* command, there exists an algorithm that will determine if a protection system with initial state $X_0$ is safe with respect to right $r$.

- Proof: determine minimum commands $k$ to leak
  - Delete/destroy: Can’t leak (or be detected)
  - Create/enter: new subjects/objects “equal”, so treat all new subjects as one
  - If $n$ rights, leak possible, must be able to leak in $n(|S_0|+1)(|O_0|+1)+1$ commands

- Enumerate all possible to decide

---

Take-Grant Protection Model

- System is directed graph
  - Subject: \( \bullet \)
  - Object: \( \bigcirc \)
  - (labeled) edge: \( \{ \text{rights} \} \)
- Take rule: if $t \in \gamma$, $\alpha \subseteq \beta$, can add transitive edge
- Grant rule: if $g \in \zeta$, $\alpha \subseteq \gamma$, can add (grant) edge between recipients
- Create, Remove rules
Theorem: Can_share(α,x,y,G₀)

• Can_share(α,x,y,G₀) iff there is an α edge from x to y in G₀ or if:
  – ∃ a vertex s ∈ G₀ with an s to y α edge,
  – ∃ a subject x' such that x'=x or x' initially spans to x,
  – ∃ a subject s' such that s'=s or s' terminally spans to s, and
  – ∃ islands I₁, ..., Iₙ such that x' ∈ I₁, s' ∈ Iₙ, and there is a bridge from Iₖ to Iₖ₊₁

• Proof: If: x' grants to x, s' takes from s, otherwise as with subjects
  – Only if: as before, plus object can’t give (receive) a right unless someone can take (grant) it

• Corollary: There is an O(|V|+|E|) algorithm to test can_share

Theorem: When Theft Possible

• Can_steal(α,x,y,G₀) iff there is no α edge from x to y in G₀ and ∃ G₁, ..., Gₙ s. t.:
  – There is no α edge from x to y in G₀,
  – ∃ subject x' such that x'=x or x' initially spans to x, and
  – ∃ s with α edge to y in G₀ and can_share(t,x',s,G₀)

• Proof:
  – ⇒: (easy – build path)
  – ⇐: Assume can_steal:
    • No α edge from definition.
    • Can_share(α,x,y,G₀) from definition: α from x to y in Gₙ
    • s exists from can_share and Monday’s theorem
    • Can_share(t,x',s,G₀): s can’t grant α (definition), someone else must get α from s, show that this can only be accomplished with take rule
Schematic Protection Model

• Key idea: Protection Type $\tau$
  – Label that determines how control rights affect an entity
  – Take-Grant: subject and object are different protection types
  – Unix file system: File, Directory, ???

• Ticket: Describes a set of rights
  – Entity has set $\text{dom}(X)$ of tickets $Y/z$ describing $X$'s rights $z$ over entities $Y$

• Inert right vs. Control right
  – Inert right doesn’t affect protection state

Transferring Rights

• Link predicate: $\text{link}_i(X,Y)$
  – conjunction or disjunction of
    • $X/z \in \text{dom}(X)$, $X/z \in \text{dom}(Y)$
    • $Y/z \in \text{dom}(X)$, $Y/z \in \text{dom}(Y)$
    • true
  – Determines if $X$ and $Y$ “connected” to transfer right
  – Example: $\text{link}(X,Y) = Y/g \in \text{dom}(X) \lor X/t \in \text{dom}(Y)$

• Filter function: conditions on transfer
• Copy $X/r:c$ from $Y$ to $Z$ allowed iff $\exists i$ such that:
  – $X/rc \in \text{dom}(Y)$
  – $\text{link}_i(Y,Z)$
  – $\tau(X)/r:c \in \text{filter}_i(\tau(Y), \tau(Z))$
Safety Analysis in SPM

- Idea: derive *maximal state* where changes don’t affect analysis
  - Similar to determining max flow
- Theorems:
  - A maximal state exists for every system
  - If parent gives child only rights parent has (conditions somewhat more complex), can easily derive maximal state

Typed Access Matrix Model

- Finite set $T$ of types ($TS \subseteq T$ for subjects)
- Protection State: $(S, O, \tau, A)$
  - $\tau: O \rightarrow T$ is a type function
  - Operations same as Harrison-Ruzzo-Ullman except create adds type
- $\tau$ is child type iff command creates create subject/object of type $\tau$ (otherwise parent)
- If parent/child graph from all commands acyclic, then:
  - Safety is decidable
  - Safety is NP-Hard
  - Safety is polynomial if all commands limited to three parameters
Bell-LaPadula: Basics

- Mandatory access control (Security Level)
  - Subject has clearance \( L(S) = l_s \)
  - Object has classification \( L(O) = l_o \)
  - Clearance/Classification ordered
    - \( l_i < l_{i+1} \)
- Discretionary access control
  - Matrix: Subject has read (write) on Object
- Need both to perform operation

Access Rules

- Simple Security Condition: \( S \) can read \( O \) if and only if
  - \( S \) dom \( O \)
  - \( S \) has discretionary read access to \( O \)
- \(*\)-Property: \( S \) can write \( O \) if and only if
  - \( O \) dom \( S \)
  - \( S \) has discretionary write access to \( O \)
- Secure system: One with above properties
- Theorem: Let \( \Sigma \) be a system with secure initial state \( \sigma_0 \), \( T \) be a set of state transformations
  - If every element of \( T \) follows rules, every state \( \sigma_i \) secure
Integrity Policy

• Principles:
  – Separation of Duty: Single person can’t mess up the system
    • No coding on live system
  – Separation of function
    • No development on production data
  – Auditing
    • Controlled/audited process for updating code on production system
• This enables validated code to maintain integrity
  – But how do we ensure we’ve accomplished these?
  – Is this overkill?

Policies

• Ring Policy
  – \( sr \leq o \leq i(o) \leq i(s) \)
  – \( s \times s \leftrightarrow i(s) \leq i(s_1) \)
• Low-Water-Mark Policy
  – \( sr \rightarrow i(s) = \min(i(s), i(o)) \)
  – \( s \rightarrow i(o) \leq i(s) \)
  – \( s_1 \times s_2 \leftrightarrow i(s_2) \leq i(s_1) \)
• Biba’s Model: Strict Integrity Policy
  – \( sr \leftrightarrow i(s) \leq i(o) \)
  – \( s \rightarrow i(o) \leq i(s) \)
  – \( s_1 \times s_2 \leftrightarrow i(s_2) \leq i(s_1) \)
• Theorem for induction similar to Bell-LaPadula
Domain-specific Policy Models

- Military Confidentiality
  - Bell-LaPadula
- Database Integrity
  - Clark/Wilson
- Corporate Anti-Trust
  - Chinese Wall
- Clinical Information Systems
- Others?

What is Consistent?

- Principle of autonomy:
  - Access allowed by security policy of a component must be allowed by composition
- Principle of security:
  - Access denied by security policy of a component must be denied by composition
- Must prove new “composed” policy meets these principles
Information Flow

• Information Flow: Where information can move in the system
• How does this relate to confidentiality policy?
  – Confidentiality: What subjects can see what objects
  – Flow: Controls what subjects actually see
• Variable $x$ holds information classified $S$
  – $x$, information flow class of $x$, is $S$
• Confidentiality specifies what is allowed
• Information flow describes how this is enforced

Formal Definition

• Problem: capturing all information flow
  – Files
  – Memory
  – Page faults
  – CPU use
  – ?
• Definition: Based on entropy
  – Flow from $x$ to $y$ (times $s$ to $t$) if $H(x_s | y_t) < H(x_s | y_s)$
How do we Manage Information Flow?

- Information flow policy
  - Captures security levels
  - Often based on confinement
  - Principles: Reflexivity, transitivity
- Compiler-based mechanisms
  - Track potential flow
  - Enforce legality of flows
- Execution-based mechanisms
  - Track flow at runtime
  - Validate correct

Confinement

- Confinement Problem
  - Prevent a server from leaking confidential information
- Covert Channel
  - Path of communication not designed as communication path
- Transitive Confinement
  - If a confined process invokes a second process, invokee must be as confined as invoker
Isolation

- Virtual machine
  - Simulates hardware of an (abstract?) machine
  - Process confined to virtual machine
    - Simulator ensures confinement to VM
  - Real example: IBM VM/SP
    - Each user gets “their own” IBM 370
- Sandbox
  - Environment where actions restricted to those allowed by policy

Covert Channels

- Storage channel
  - Uses attribute of shared resource
- Timing channel
  - Uses temporal/ordering relationship of access to shared resource
- Noise in covert channel
  - Noiseless: Resource only available to sender/receiver
  - Noisy: Other subjects can affect resource
Modeling Covert Channels

- Noninterference
  - Bell-LaPadula approach
  - All shared resources modeled as subjects/objects
  - Let $\sigma \in \Sigma$ be states. Noninterference secure if $\forall s$ at level $l(s) \ni \exists: \Sigma \times \Sigma$ such that
    - $\sigma_1 \equiv \sigma_2 \Rightarrow \text{view}(\sigma_1) = \text{view}(\sigma_2)$
    - $\sigma_1 \equiv \sigma_2 \Rightarrow \text{execution}(i, \sigma_1) \equiv \text{execution}(i, \sigma_2)$
    - if $i$ only contains instructions from subjects dominating $s$, $\text{view(}\text{execution}(i, \sigma)) = \text{view}(\sigma)$

- Information Flow analysis
  - Again model all shared resources

What is Authentication?

- Authentication: Binding of identity to subject
- How do we do it?
  - Entity knows something
    - Passwords, id numbers
  - Entity has something
    - Badge, smart card
  - Entity is something
    - Biometrics
  - Entity is someplace
    - Source IP, restricted area terminal
Authentication System: Formal Definition

- **A**: set of *authentication information*
  - used by entities

- **C**: set of *complementary information*
  - used by system to validate authentication information

- **F**: set of *complementation functions*
  - \( f : A \rightarrow C \)
  - Generate appropriate \( c \in C \) given \( a \in A \)

- **L**: set of *authentication functions*
  - \( l : A \times C \rightarrow \{ \text{true, false} \} \)
  - verify identity

- **S**: set of *selection functions*
  - \( s : ? \rightarrow A \)
  - Generate/alter \( A \) and \( C \)

Authentication Systems

- **Passwords**
  - Information known to subject
  - Complementation Function
    - Null – requires that \( c \) be protected
    - One-way hash – function such that
      - \( f(a) \) easy to compute
      - \( f^{-1}(c) \) difficult to compute

- **Challenge-Response**
  - Pass algorithm
    - authenticator sends message \( m \)
    - subject responds with \( a(m) \)
  - One-time password
    - \( a \) changes after use
    - Why is this challenge-response?

- **Location**
  - Secured area
  - Limited geographic area
  - Movement tracking

- **Biometrics**
Authentication vs. Identity

• Authentication: Binding of identity to subject
  – We know how
  – We’ve discussed subjects
  – But what precisely is identity?
• Principal: Unique Entity
  – Subject
  – Object
• Identity: Specifies a principal

Representing Identity

• Randomly chosen: not useful to humans
• User-chosen: probably not unique
  – At least globally
• Hierarchical: Disambiguate based on levels
  – File systems
  – X.500: Distinguished Names
    • /O=Purdue University/OU=Computer Sciences/CN=clifton
  – Aliases
    • /O=Purdue University/OU=ITaP/CN=cclifton
Validating Identity

- Authentication: Does subject match purported identity?
- Problem: Does identity match principal?
- Solution: certificates
  - Certificate: Identity validated to belong to known principal
  - Certification Authority: Certificate Issuer
    - Authentication Policy: describes authentication required to ensure principal correct
    - Issuance policy: Who certificates will be issued to
  - CA is trusted

Anonymity

- What if identity not needed?
  - Web browsing
  - Complaints about assignments
- Removing identity not as easy as it sounds
  - I can send email without my userid
  - But it still traces back to my machine
- Solution: anonymizer
  - Strips identity from message
  - Replaces with (generated) id
  - Send to original destination
  - Response: map generated id back to original identity
Cryptography

• Encryption system $E_k(s)$
  – over all choices of $k$, should form uniform distribution
  – Thus “appears” independent of $s$
• Decryption: $D_k(E_k(s)) = s$
• Strength: Given $E_k(s)$, finding $k$ or $s$ difficult
  – Better: given $s$, $E$, $E_k(s)$, learning $k$ hard
  – choosing $k$, computing $D_k$, $E_k$ must be easy
• Public key: $p, r$ such that $D_r(E_p(s)) = s$
  – Similar notions of strength
Secure System: Design Principles

• Least Privilege
• Fail-Safe Defaults
• Economy of Mechanism
• Complete Mediation
• Open Design
• Separation of Privilege
• Least Common Mechanism
• Psychological Acceptability

Assurance: Confidence a system meets security requirements

• Trusted System: Evaluated / passed in terms of well-defined requirements, evaluation methods
• High-Assurance Development Methodologies Control:
  – Requirements definitions, omissions, mistakes
  – System design flaws
  – Hardware implementation flaws
  – Software implementation errors
  – System use/operation errors
  – Willful system misuse
  – Hardware malfunction
  – Natural / environmental effects
  – Evolution/maintenance/upgrades/decommission
Assurance: Where?

- Design Assurance
  - Evidence that Design meets Security Policy
- Implementation Assurance
  - Evidence that the implementation meets the design
- Operational/Administrative Assurance
  - Evidence that policy requirements maintained in operation
    - Best: evidence that system can’t enter non-secure state

Assurance: How?

- Mistakes will be made
  - Must they lead to security violations?
- Solution: Risk Mitigation
- Definitions:
  - Threat: Potential occurrence leading to undesirable consequences
  - Vulnerability: Weakness enabling threat
  - Exploit: Method for Threat to use Vulnerability
- All must occur for a violation to happen
Add-On Security

- Implement security in separate module
  - Easier to validate
  - But may be hard to enforce
- Reference Monitor
  - Abstract machine that mediates all accesses
  - Implement with Reference Validation Mechanism
- Security Kernel: Implements reference Monitor
- Trusted Computing Base: subset of system that enforces security policy
  - Demands extra protection

Evaluating Assurance

- How do we gather *evidence* that system meets security requirements?
- Process-based techniques: Was system constructed using proper methods?
  - SEI CMM
  - ISO 9000
- System Evaluation
  - Requirements Tracing
  - Representation Correspondence
  - Reviews
  - Formal Methods
Implementation Management

- Assume a secure design
  - How to ensure implementation will be secure?
- Constrained Implementation Environment
  - Strong typing
  - Built-in buffer checks
  - Virtual machines
- Coding Standards
  - Restrict how language is used
  - Meeting standards eliminates use of “unsafe” features

System Evaluation

- Requirements Tracing
  - Track requirement to mechanism
- Representation Correspondence
  - Requirements tracing between levels
- Validating Correctness:
  - Informal arguments
  - Formal verification
- Reviews
  - Formal process of “passing” on design/implementation/etc.
Process Guidance Working Group Test Model

- Test Matrix: Maps requirements to lower levels
  - At lowest level, test assertion
  - Used to develop test cases
- Divides checks into six areas
  - Discretionary Access Control
  - Privileges
  - Identification and Authorization
  - Object Reuse
  - Audit
  - System Architecture Constraints

Operation/Maintenance

- Fixes / maintenance
  - Hot fix: quick solution
    - Possibly security testing only
    - May limit functionality
  - Regular fix: more thorough testing
    - Reintroduce functionality while maintaining security
- Procedures to track flaws
  - Reporting
  - Test to detect flaw
  - Regression test: ensure flaw not “unfixed”
Formal Verification: Components

- Formal Specification defined in unambiguous (mathematical) language
  - Example: security policy models
- Implementation Language
  - Generally somewhat constrained
- Formal Semantics relating the two
- Methodology to ensure implementation ensures specifications met

Verification Methodologies

- Proof based vs. model based
  - Proof: Formula define premises / conclusions
    - Proof shows how to reach conclusions from premises
  - Model-based: Premises and conclusions have compatible truth tables
- Full vs. property verification
  - Does methodology model full system?
  - Or just prove certain key properties?
- Automation – may be manual or have tool support
Formal Evaluation

- Method to achieve Trust
  - Not a guarantee of security
- Evaluation methodology includes:
  - Security requirements
  - Assurance requirements showing how to establish security requirements met
  - Procedures to demonstrate system meets requirements
    - Metrics for results
- Examples: TCSEC (Orange Book), ITSEC, CC

Later Standards

- CTCPEC – Canada
- ITSEC – European Standard
  - Did not define criteria
  - Levels correspond to strength of evaluation
  - Includes code evaluation, development methodology requirements
  - Known vulnerability analysis
- CISR: Commercial outgrowth of TCSEC
- FC: Modernization of TCSEC
- FIPS 140: Cryptographic module validation
- Common Criteria: International Standard
- SSE-CMM: Evaluates developer, not product
• Replaced TCSEC, ITSEC
• CC Documents
  – Functional requirements
  – Assurance requirements
  – Evaluation Assurance Levels
• CC Evaluation Methodology
  – Detailed process model for each level
• National Scheme

Common Criteria: Protection Profile

Domain-specific set of security requirements
• Narrative Overview
• Domain description
• Security Environment (threats, overall policies)
• Security Objectives: System, Environment
• IT Security Requirements
  – Functional drawn from CC set
  – Assurance level
• Rationale for objectives and requirements
Common Criteria: Security Target

Specific requirements used to evaluate system

- Narrative introduction
- Environment
- Security Objectives
  - How met
- Security Requirements
  - Environment and system
  - Drawn from CC set
- Mapping of Function to Requirements
- Claims of Conformance to Protection Profile

Common Criteria: Functional Requirements

- 362 page document
- 17 Classes
  - Audit, Communication, Cryptography, User data protection, ID/authentication, Management, Privacy, Protection of Security Functions, Resource Utilization, Access, Trusted paths
- Several families per class
- Lattice of components in family
Common Criteria: Assurance Requirements

- 232 page document
- 8 Classes
  - Protection Profile Evaluation, Security Target Evaluation
  - Configuration management, Delivery and operation, Development, Guidance, Life cycle, Tests, Vulnerability assessment
  - Maintenance, Composition
- Several families per class
- Lattice of components in family

Common Criteria: Evaluation Assurance Levels

1. Functionally tested
2. Structurally tested
3. Methodically tested and checked
4. Methodically designed, tested, and reviewed
5. Semiformally designed and tested
6. Semiformally verified design and tested
7. Formally verified design and tested
Audit

An a-posteriori technique to identify security violations

• What information do we need?
  – After the fact – current state of system isn’t enough
  – **logging**

• How do we perform an Audit?
  – Audit methodology

• What do we do with the results?

Logging

• Goal: Record all information that might be needed for an audit
  – Authentication attempts
  – Access to trusted resources

• Log must enable detection of security violations
  – Is this enough?

• Secure system **prevents** security violations

• Trusted components: those that can violate security
  – Assumptions made to justify system secure

• Log actions by trusted components
  – Change in security level
  – Operations performed while not at maximum level
Audit

- Detect security violation
  - State-based auditing: identify if state at prior time is valid
  - Transition-based auditing: Identify if prior transition would lead to unauthorized state
- Detect attempts to breach security
  - Not necessarily violations

What is Malicious Code?

- Set of instructions designed to violate security policy
- Generally relies on “legal” operations
  - Authorized user could perform operations without violating policy
  - Malicious code “mimics” authorized user
- Types of Malicious Code
  - Trojan Horse - Trick user into executing malicious code
  - Virus - Replicates into fixed set of files
  - Worm - Copies itself from computer to computer
Detection

- Undecidable if code contains a virus
- Heuristics
  - Signature-based antivirus
  - Checksum
  - Validate action against specification
  - Proof-carrying code
  - Statistical Methods

Defense

- Clear distinction between data and executable
  - Virus must write to program
  - Must execute to spread/act
  - Auditable action required to change data to executable
- Information Flow
  - Malicious code usurps authority of user
  - Limit information flow between users
  - Limits spread of virus
  - Problem: Tracking information flow
- Least Privilege
  - Programs run with minimal needed privilege
  - Example: Limit file types accessible by a program
Vulnerability Analysis

- Vulnerability: Lapse in enforcement enabling violation of security policy
- Exploit: Use of vulnerability to violate policy
- How it’s done:
  - System Verification
  - Penetration testing

Penetration Testing

- Test complete system
  - Attempt to violate stated policy
- Typical approach: Red Team, Blue Team
  - Red team attempts to discover vulnerabilities
  - Blue team simulates normal administration
  - White team injects workload, captures results
- Types of Penetration Testing
  - Black Box - External attacker has no knowledge of target system
  - Red team provided with limited access to system
    - Goal is to gain normal or elevated access
  - Internal attacker
    - Red team provided with authorized user access
    - Goal is to elevate privilege / violate policy
Red Team Approach

- Information gathering
  - Examine design, environment
- Flaw hypothesis
  - Predict likely vulnerabilities
- Flaw testing
  - Determine where vulnerabilities exist
- Flaw generalization
  - Attempt to broaden discovered flaws
- Flaw elimination
  - Suggest means to eliminate flaw

Vulnerability Classification

- Goal: describe spectrum of possible flaws
- How do we classify?
  - By how they are exploited?
  - By where they are found?
  - By the nature of the vulnerability?
Research Into Secure Operating Systems (RISOS)

- Seven Classes
  1. Incomplete parameter validation
  2. Inconsistent parameter validation
  3. Implicit sharing of privileged / confidential data
  4. Asynchronous validation / inadequate serialization
  5. Inadequate identification / authentication / authorization
  6. Violvable prohibition / limit
  7. Exploitable logic error
- Evaluated several operating systems

Protection Analysis Model

- Pattern-directed protection evaluation
  - Methodology for finding vulnerabilities
- Applied to several operating systems
  - Discovered previously unknown vulnerabilities
- Resulted in two-level hierarchy of vulnerability classes
  - Ten classes in all
Others

- NRL Taxonomy: Three classification schemes
  - Type of flaw
  - When was it “created”
  - Where is it
- Aslam’s Model
  - Attempts to classify faults unambiguously
    - Decision procedure to classify faults
  - Coding Faults
    - Synchronization errors
    - Condition validation errors
  - Emergent Faults
    - Configuration errors
    - Environment Faults

Flow-based Penetration Analysis
(Gupta and Gligor ’91)

- Goal: Systematic approach to penetration analysis
  - Verifiable properties
  - Amenable to automation
- Assumes set of design properties will produce secure system
  - System Isolation (Tamperproof)
  - System Noncircumventability (complete mediation)
  - Consistency of Validation Checks
  - Elimination of Undesirable System/User Dependencies
- Captures properties using state-transition mode
  - Entity may be altered/viewed/invoked only if preconditions validated in atomic sequence
Intrusion Detection/Response

• Denning: Systems under attack fail to meet one or more of the following characteristics:
  1. Actions of users/processes conform to statistically predictable patterns
  2. Actions of users/processes do not include sequences of commands to subvert security policy
  3. Actions of processes conform to specifications describing allowable actions

Intrusion Detection

• Idea: Attack can be discovered by one of the above being violated
  – Problem: Definitions hard to make precise

• Practical goal of intrusion detection systems:
  – Detect a wide variety of intrusions
  – Detect in a timely fashion
  – Present in a useful manner
  – Be (sufficiently) accurate
IDS Types

- **Anomaly Detection**
  - Compare characteristics of system with expected values
  - Threshold metric: when statistics deviate from normal by threshold, sound alarm
  - Statistical moments: based on mean/standard deviation of observations
  - Markov model: based on state, expected likelihood of transition to new states

- **Misuse Detection**
  - Does sequence of instructions violate security policy?
  - Solution: capture *known* violating sequences
  - Alternate solution: State modeling
    - Known "bad" state transition from attack

IDS Architecture

- **Similar to Audit system**
  - Log events
  - Analyze log
  - Difference: happens real-time

- **Host based IDS**
  - watches events on the host
  - Often uses existing audit logs

- **Network-based IDS**
  - Packet sniffing
  - Firewall logs
**Intrusion Response**

- **Incident Prevention**
  - Stop attack before it succeeds
  - Measures to detect attacker
    - Honeypot / Jailing

- **Intrusion handling**
  - Contain attack
  - Eradicate attack
  - Recover to secure state
  - Punish attacker

**Network Security**

```
Intranet
  __________
  |          |
  |          |
  |          |
  |          |
  |_________|
    |         |
    |         |
    |
    |
  File Server
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  Web Server
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  Mail server
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  DNS (internal)
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  User machines

Demilitarized Zone (DMZ)
  __________
  |          |
  |          |
  |          |
  |          |
  |_________|
    |         |
    |         |
    |
    |
  Mail forwarding
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  Web Server
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  DNS (DMZ)
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  Firewall
  |         |
  |         |
  |         |
  |________|
    |         |
    |         |
    |
    |
  Internet
```

©2010 Prof. Chris Clifton
Typical network: Terms

• Network Regions
  – Internet
  – Intranet
  – DMZ
• Network Boundaries
  – Firewall
    • Filtering firewall: Based on packet headers
    • Audit mechanism
  – Proxy
    • Proxy firewall: Gives external view that hides intranet

Issues

• IP: Intranet hidden from outside world
  – Internal addresses can be real
    • Proxy maps between real address and firewall
    • Network Address Translation Protocol maps internal to assigned address
• Mail Forwarding
  – Hide internal addresses
  – Map incoming mail to “real” server
  – Additional incoming/outgoing checks
Network Security: Tools

- Confidentiality
  - Encryption
- Integrity
  - Digital Signatures
  - Retransmission
  - Order?
- Availability
  - Quality of Service
- Distributed Authentication
- Synchronization and Commit

Distributed Authentication: Kerberos

- Kerberos developed at MIT in the 1980s
  - Project Athena: clusters of publicly available computers for student/faculty use
  - Shared file service – log in anywhere
  - Problem: how to ensure user logging in at A authorized to use resources at B?
- Solution: ticket as credential
  - Ticket server
  - Client
  - Client address
  - Valid time
  - Session key
  
  Encrypted with ticket server’s key
Multiple Domains: **X.500**

- **Goal:** Global “white pages”
  - Lookup anyone, anywhere
  - Developed by Telecommunications Industry
  - ISO standard directory for OSI networks
- **Idea:** Distributed Directory
  - Application uses Directory User Agent to access a Directory Access Point

**Attacks and Defense**

- **Confidentiality on the network manageable**
  - Encryption to protect transmission
  - Public key cryptography / key management to verify recipient
- **Integrity reducible to single system**
  - Digital signatures verify source
  - Commit protocols handle network failure
- **What about Availability?**
Network Attacks

• Flooding
  – Overwhelm TCP stack on target machine
  – Prevents legitimate connections

• Routing
  – Misdirect traffic

• Spoofing
  – Imitate legitimate source

Test Taking Hints

• Exam will be open book/note
  – Same as the midterm

• Expect a bit more writing
  – Still relatively short answers

• Watch the time
  – Don't be nervous…

• See old copies of my exams (and solutions) at old course pages
  – CS 541
  – CS 603?