February 4, 2014

- Assurance
  - Requirements Assurance
  - Justifying Requirements
  - Design Assurance
  - Documentation and Specification
  - Reviews of Assurance Evidence

Administration

- Discussion section tomorrow, none next week
- Project presentations *next* week
  - On both Tuesday Feb 11 and Thursday Feb 13
  - Plan on 10 min *maximum* (including Q&A)
  - Email me if you prefer Tuesday or Thursday

More Administration

- People who are scheduled to present papers this week, please come see me after class
  - Siyuan Gu
  - Pengfei Hu
  - Zhicheng Yang

Requirements Assurance

- *Specification* describes of characteristics of computer system or program
- *Security specification* specifies desired security properties
- Must be clear, complete, unambiguous
  - Something like "meets C2 security requirements" not good: what *are* those requirements (actually, 34 of them!)

Example

- "Users of the system must be identified and authenticated" is ambiguous
  - Type of id required—driver's license, token?
  - What is to be authenticated—user, representation of identity, system?
  - Who is to do the authentication—system, guard?

Example

- "Users of the system must be identified to the system and must have that identification authenticated by the system" is less ambiguous
  - Under what conditions must the user be identified to the system—at login, time of day, or something else?

Example

- "Users of the system must be identified to the system and must have that identification authenticated by the system before the system performs any functions on behalf of that identity"
  - Type of identification is user name
  - User identification (name) to be authenticated
  - System to authenticate
  - Authentication to be done at login (before system performs any action on behalf of user)

Methods of Definition

- Extract applicable requirements from existing security standards
  - Tend to be semiformal
- Combine results of threat analysis with components of existing policies to create a new policy
- Map the system to existing model
  - If model appropriate, creating a mapping from model to system may be cheaper than requirements analysis

Example

- System X: UNIX system with MAC based on Bell-LaPadula Model
  - Mapping constructed in series of stages
  - Auditing also required

Example Stage 1

- Map elements, state variables of BLP to entities in System X
  - Subject set *S* in BLP → set of processes
  - Object set *O* in BLP → set of inode objects, IPC objects, mail messages, processes as destinations, passive entities
  - Right set *P* in BLP → set of rights of system functions
    - Functions that create entities, write entities, have write w
    - Functions that read entities have right r
    - Functions that execute, search entities have right r

Example Stage 1

- Access set b in BLP → types of access
  - Subjects can use rights r, w, a to access inode objects.
- Access control matrix a for current state in BLP → current state of mandatory and discretionary controls
- Functions f_s, f_o, and f_c in BLP → three functions
  - $f(s)$ is the maximum security level of subject s
  - *current-level*(s) is current security level of subject s
  - $f(o)$ is the security level of object o

Example Stage 1

- Hierarchy H in BLP → differently for different objects
  - Inode objects are hierarchical trees represented by the file system hierarchy
  - Other object types map to discrete points in the hierarchy

Example Stage 2

- Define BLP properties in language of System X and show each property is consistent with BLP
  - MAC property of BLP → user having over an object:
    - read access iff user's clearance dominates object's classification
    - write access over an object iff object's classification dominates user's clearance.
  - DAC property of BLP → user having access to object iff owner of object has explicitly granted that user access to object

Example Stage 2

- Label inheritance, user level changes specific to System X
  - Security level of newly created object inherited from creating subject
  - Security level of initial process at user login, security level of initial process after user level change, bounded by security level range defined for that user and for the terminal
  - Security level of newly spawned process inherited from parent, except for first process after a user level change
  - When user's level raised, child process does not inherit write access to objects opened by parent
  - When user's level lowered, all processes, accesses associated with higher privilege terminated

Example Stage 2

- Reclassification property of System X
  - Specially trusted users allowed to downgrade objects they own within constraints of user's authorizations.
- System X property of owner/group transfer allows ownership or group membership of process to be transferred to another user or group
- Status property is property of System X
  - Restricts visibility of status information available to users when they use standard System X set of commands

Example Stage 3

- Designers define System X rules by mapping System X system calls, commands, and functions to BLP rules
  - Simple security condition, *-property, and discretionary security property interpreted for each type of access
  - From these interpretations, designers can extract specific requirements for specific accesses to particular types of objects.

Example Stage 4

- Designers show System X rules preserve security properties
  - Show that the rules enforce the properties directly; or
  - Map the rules directly to a BLP rule or a sequence of BLP rules
    - 9 rules about current access
    - 5 rules about functions and security levels
    - 8 access permission rules
    - 8 more rules about subjects and objects
  - Designers must show that each rule is consistent with actions of System X.

Justifying Requirements

- Show policy complete and consistent
- Example: ITSEC suitability analysis
  - Map threats to requirements and assumptions
  - Describe how references address threat

Example: System Y Evaluation

- Threat T1: A person not authorized to use the system gains access to the system and its facilities by impersonating an authorized user.
  - Requirement IA1: A user is permitted to begin a user session only if the user presents a valid unique identifier to the system and if the claimed identity of the user is authenticated by the system by authenticating the supplied password.
  - Requirement IA2: Before the first user/system interaction in a session, successful identification and authentication of the user take place.

System Y Assumptions

- Assumption A1: The product must be configured such that only the approved group of users has physical access to the system.
- Assumption A2: Only authorized users may physically remove from the system the media on which authentication data is stored.
- Assumption A3: Users must not disclose their passwords to other individuals.
- Assumption A4: Passwords generated by the administrator shall be distributed in a secure manner.

System Y Mapping

Threat	Security Target Reference
T1	IA1, IA2, A1, A2, A3, A4

System Y Justifications

1. The referenced requirements and assumptions guard against unauthorized access. Assumption A1 restricts physical access to the system to those authorized to use it. Requirement IA1 requires all users to supply a valid identity and confirming password. Requirement IA2 ensures that requirement IA1 cannot be bypassed.

System Y Justifications

2. The referenced assumptions prevent unauthorized users from gaining access by using a valid user's identity and password. Assumption A3 ensures that users keep their passwords secret. Assumption A4 prevents unauthorized users from intercepting new passwords when those passwords are distributed to users. Finally, assumption A2 prevents unauthorized access to authentication information stored on removable media.

These justifications provide an informal basis for asserting that, if the assumptions hold and the requirements are met, the threat is adequately handled.

Design Assurance

- Process of establishing that design of system sufficient to enforce security requirements
  - Specify requirements (see above)
  - Specify system design
  - Examine how well design meets requirements

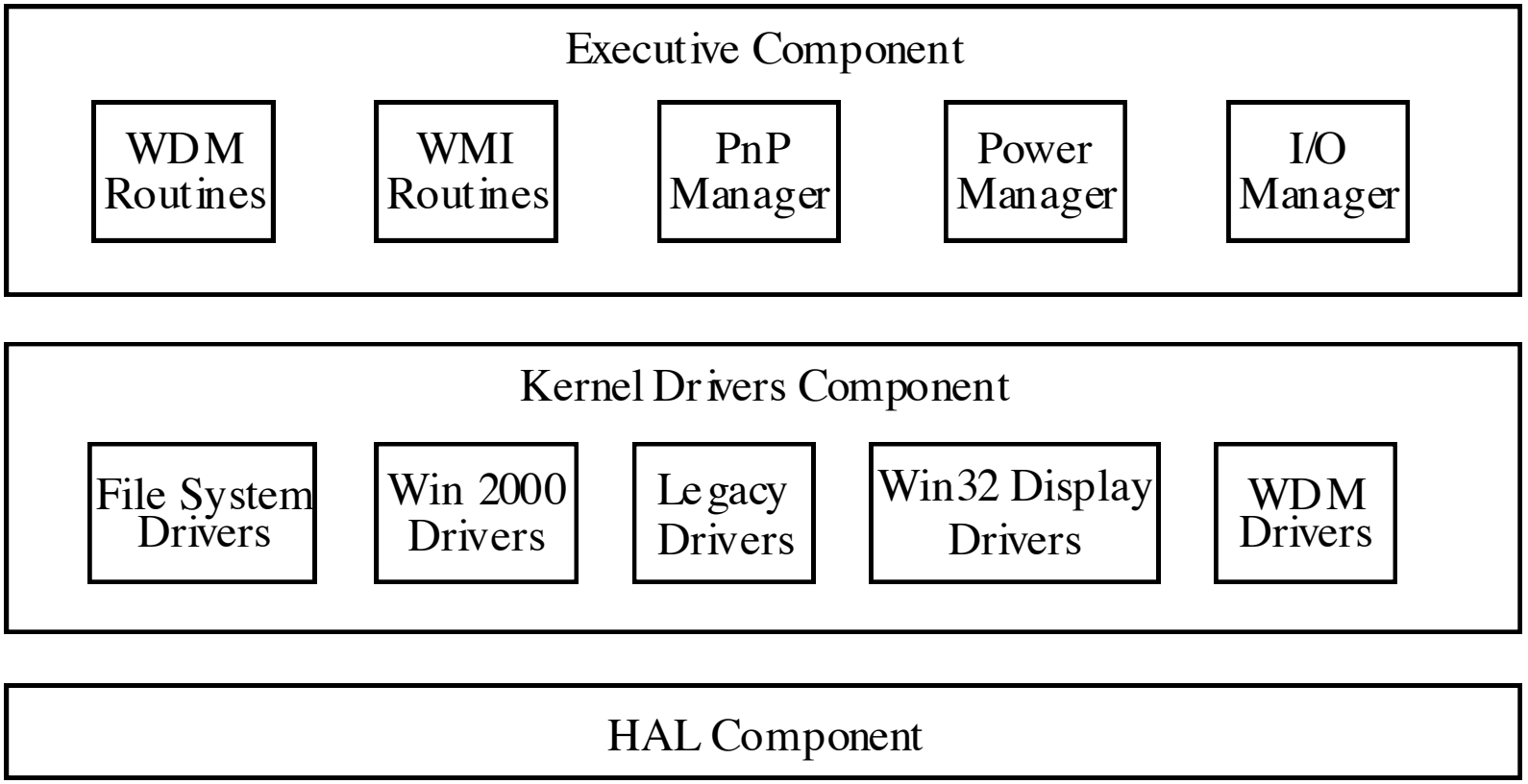
Design Techniques

- Modularity
  - Makes system design easier to analyze
  - RVM: functions not related to security distinct from modules supporting security functionality
- Layering
  - Makes system easier to understand
  - Supports information hiding

Layering

- Develop specifications at each layer of abstraction
  - *subsystem* or *component*: special-purpose division of a larger entity
    - Example: for OS, memory manager, process manager; Web store: credit card handlers
  - *subcomponent*: part of a component
    - Example: I/O component has I/O managers and I/O drivers as subcomponents
  - *module*: set of related functions, data structures

Example: Windows 2000 I/O System



Design Document Contents

- Provide basis for analysis
  - informal, semiformal, formal
- Must include:
  - *Security functions*: high-level descriptions of functions that enforce security and overview of protection approach
  - *External interfaces*: interfaces visible to users, how the security enforcement functions constrain them, and what the constraints and effects should be
  - *Internal design*: Design descriptions addressing the architecture in terms of the next layer of decomposition; also, for each module, identifies and describes all interfaces and data structures

Security Functions

- *Security functions summary specification* identifies high-level security functions defined for the system
  1. *Description of individual security functions*, complete enough to show the intent of the function; tie to requirements
  2. *Overview of set of security functions* describing how security functions work together to satisfy security requirements
  3. *Mapping to requirements*, specifying mapping between security functions and security requirements.

External Interface

- High-level description of external interfaces to system, component, subcomponent, or module
  1. *Component overview* identifying the component, its parent, how the component fits into the design
  2. *Data descriptions* identifying data types and structures needed to support the external interface descriptions specific to this component, and security issues or protection requirements relevant to data structures.

External Interface

- High-level description of external interfaces to system, component, subcomponent, or module
  3. *Interface descriptions* including commands, system calls, library calls, functions, and application program interfaces as well as exception conditions and effects

Example

- Routine for error handling subsystem that adds an event to an existing log file

Interface Name

error_t add_logevent (handle_t handle, data_t event);

Input Parameters

handle valid handle returned from previous call to *open_log*

event buffer of event data with records in *logevent* format

Example

Exceptions

- Caller does not have permission to add to EVENT file.
- There is inadequate memory to add to an EVENT file.

Effects

Event is added to EVENT log.

Output Parameters

status	status_ok	/* routine completed successfully */
	no_memory	/* insufficient memory (failed) */
	permission_denied	/* no permission (failed) */

Note

add_logevent is a user-visible interface

Internal Design

- Describes internal structures and functions of components of system
  1. *Overview of the parent component*; its high-level purpose, function, security relevance
  2. *Detailed description of the component*; its features, functions, structure in terms of the subcomponents, all interfaces (noting externally visible ones), effects, exceptions, and error messages
  3. *Security relevance of the component* in terms of security issues that it and its subcomponents should address

Example: Parent Component

- Audit component is responsible for recording accurate representation of all security-relevant events in the system and ensuring that integrity and confidentiality of the records are maintained.
  - Audit view: subcomponent providing authorized users with a mechanism for viewing audit records.
  - Audit logging: subcomponent records the auditable events, as requested by the system, in the format defined by the requirements
  - Audit management: subcomponent handling administrative interface used to define what is audited.

Example: Detailed Component Description

- Audit logging subcomponent records auditable events in a secure fashion. It checks whether requested audit event meets conditions for recording.
- Subcomponent formats audit record and includes all attributes of security-relevant event; generates the audit record in the predefined format
- Audit logging subcomponent handles exception conditions
  - Error writing to the log

Example

- Audit logging subcomponent uses one global structure:

```
structure audit_config      /* defines configuration of */
                            /* which events to audit */
```

- The audit logging subcomponent has two external interfaces:

```
add_logevent()              /* log an event */
logevent()                  /* ask to log event */
```

Example: Security Relevance

- Audit logging subcomponent monitors security-relevant events and records those events matching configurable audit selection criteria
  - Security-relevant events include attempts to violate security policy, successful completion of security-relevant actions

Low-Level Design

- Focus on internal logic, data structures, interfaces; may include pseudocode
  1. *Overview*, giving the purpose of the module and its interrelations with other modules, especially dependencies on other modules
  2. *Security relevance of the module*, showing how the module addresses security issues
  3. *Individual module interfaces*, identifying all interfaces to the module, and those externally visible.

Example: Overview

- Audit logging subcomponent
  - Responsible for monitoring and recording security-relevant events
  - Depends on I/O system and process system components
- Audit management subcomponent
  - Depends on audit logging subcomponent for accurate implementation of audit parameters configured by audit management subcomponent
- All system components depend on audit logging component to produce their audit records

Example: Overview

- Audit logging subcomponent:

Variables

structure logevent_t	defines audit record
structure audit_ptr	current position in audit file
file_ptr audit_fd	file descriptor of audit file

Global structure

```
structure audit_config /* defines configuration */
                       /* of which events are to be audited */
```

External interfaces

```
add_logevent()         /* begin logging events of given type */
logevent()             /* ask to log event */
```

Example: Security Relevance

- Audit logging subcomponent monitors security-relevant events, records those events matching the configurable audit selection criteria
  - Example: attempts to violate security policy
  - Example: successful completion of security-relevant actions
- Audit logging subcomponent must ensure no audit records are lost, and are protected from tampering

Example: Individual Interfaces

- *logevent*() only external interface

  verify function parameters

  call *check_selection_parameters* to determine if system has been configured to audit event

  if *check_selection_parameters* then

    call *create_logevent*

    call *write_logevent*

    return success or error number

  else

    return success

Example: Individual Interfaces

- *add_logevent*() available only to privileged users

  verify caller has privilege/permission to use this function

  if caller does not have permission

  return permission_denied

  verify function parameters

  call *write_logevent* for each event record

  return success or error number from *write_logevent*

Internal Design

- *Introduction*: purpose, scope, target audience
- *Component overview*: identifies modules, data structures; how data is transmitted; security relevance and functionality
- *Detailed module designs*
  - *Module #1*: module's interrelations with other modules, local data structures, its control and data flows, security
    - *Interface Designs*: describes each interface
    - Interface 1a: security relevance, external visibility, purpose, effects, exceptions, error messages, and results

Example

- Windows 2000 I/O System
  - High-level design document describes I/O system as a whole
    - Necessary descriptions of executive, kernel driver, HAL
  - Describes first level of design decomposition
- Next level of decomposition
  - High-level design document for I/O file drivers
  - Internal design spec for HAL component
- Internal design specs for each subcomponent of I/O file drivers

Documentation and Specification

- Time, cost, efficiency may impact how complete set of documents prepared
- Different types of specifications
  - Modification Specifications
  - Security Specifications
  - Formal Specifications

Modification Specifications

- Used when system built from previous versions or components
  - Specifications for these versions or components
  - Specifications for changes to, additions of, and methods for deleting modules, functions, components
- Developer understands the system upon which the new system is based

Security Considerations

- Security analysis must rest on specification of current system, not previous ones or changes only
  - If modification specifications are only ones, security analysis based upon incomplete specifications
  - If previous system has full security specifications, then analysis may be complete

Security Specifications

- Used when design specifications adequate except for security issues
- Develop supplemental specifications to describe missing security functionality
  - Develop document that starts with security functions summary specification
  - Expand to address security issues of components, subcomponents, modules, functions

Example: System X

- Underlying UNIX system completely specified, including complete functional specifications and internal design specifications
  - Neither covered security well, let alone document new functionality

Example: System X

- Team supplemented existing documentation with security architecture document
  - Addresses deficiencies of existing documentation
  - Gives complete overview of each security function
  - Additional documentation describes external interface, internal design of all functions

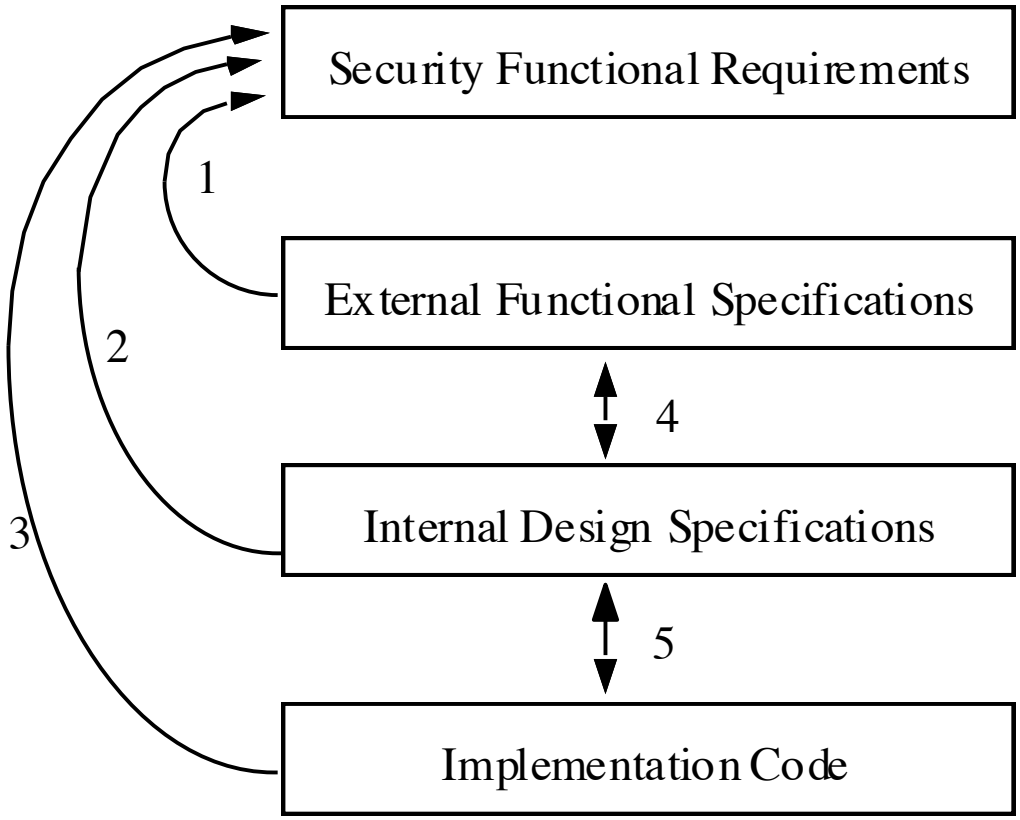
Formal Specifications

- Any specification can be formal
- Written in formal language, with well-defined syntax and sound semantics
- Supporting tools allow checking
  - Parsers
  - Theorem provers

Justifications

- Formal techniques
  - Proofs of correctness, consistency
- Informal techniques
  - Requirements tracing: showing which specific security requirements are met by parts of a specification
  - Informal correspondence: showing a specification is consistent with adjacent level of specification

Requirements and Informal Correspondence



Reviews of Assurance Evidence

- Reviewers given guidelines for review
- Other roles:
  - Scribe: takes notes
  - Moderator: controls review process
  - Reviewer: examines assurance evidence
  - Author: author of assurance evidence
  - Observer: observe process silently
- Important: managers may *only* be reviewers, and only then if their technical expertise warrants it

Setting Review Up

- Moderator manages review process
  - If not ready, moderator and author's manager discuss how to make it ready with author
  - May split it up into several reviews
  - Chooses team, defines ground rules
- Technical Review
  - Reviewers follow rules, commenting on any issues they uncover
    - May request moderator to stop review, send back to author
  - General and specific comments to author

Review Meeting

- Moderator is master of ceremonies
  - Grammatical issues presented first
  - General and specific comments next
  - Goal is to collect comments on entity, *not* to resolve differences
  - Scribes write down comments and who made it (anyone can see it, help scribe, verify comment made)

Conflict Resolution

- After meeting, scribe creates Master Comment List
  - Reviewers mark "Agree" or "Challenge"
  - All comments that everyone "Agree"s are put on Official Comment List
  - Rest must be resolved by reviewers
- Moderator, reviewers then:
  - Accept as is
  - Accept with changes on OCL
  - Reject

Conflict Resolution

- Author takes OCL, makes changes as sees fit
- Author then meets with reviewers
  - Explains how each comment made by reviewer was handled
  - All must be resolved to satisfaction of author, reviewer
- Review completed

Key Points

- Assurance critical for determining trustworthiness of systems
- Different levels of assurance, from informal evidence to rigorous mathematical evidence
- Assurance needed at all stages of system life cycle