April 28: Assurance

- Design assurance
- Documentation
- Review of assurance evidence

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

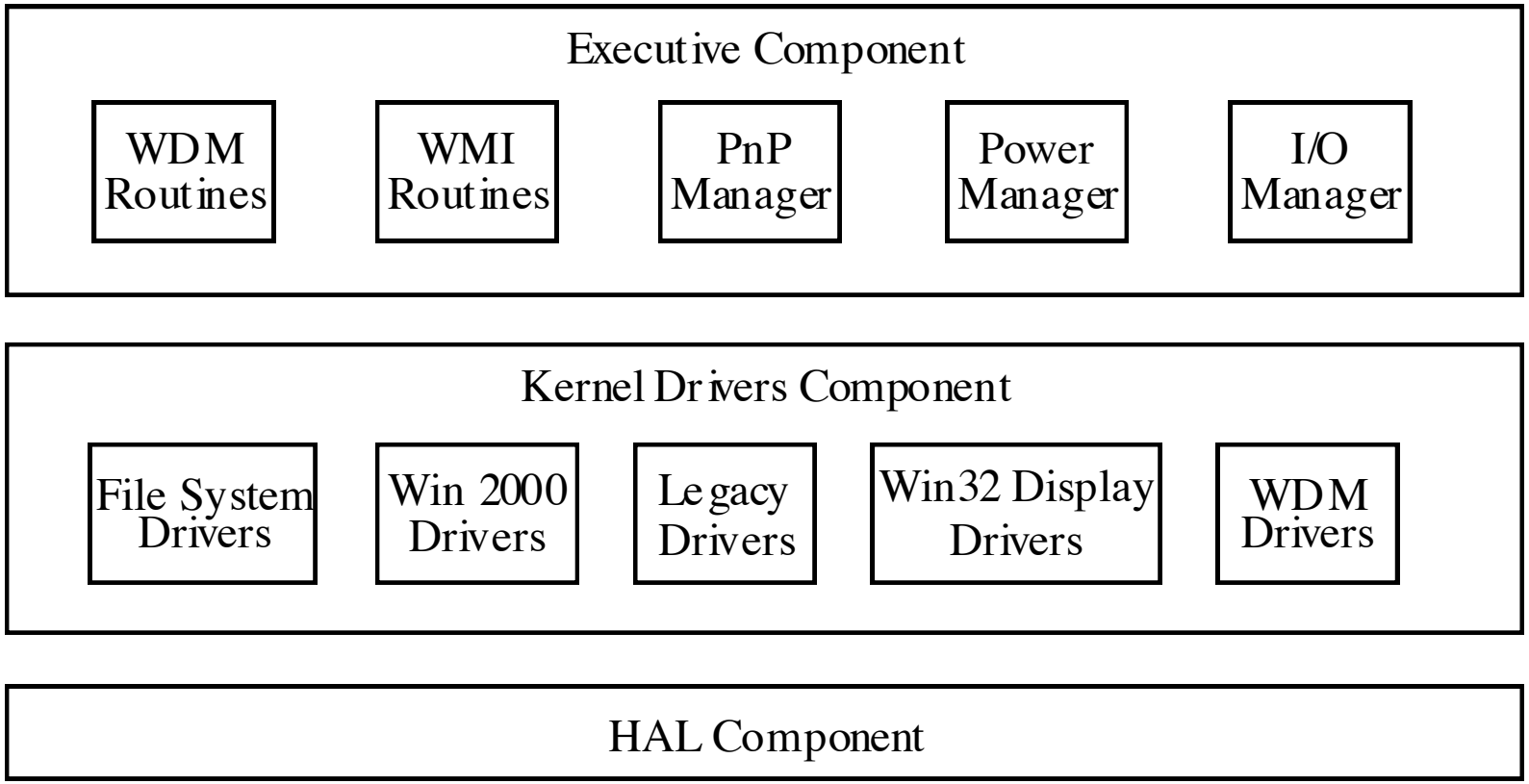
Design Techniques

- 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

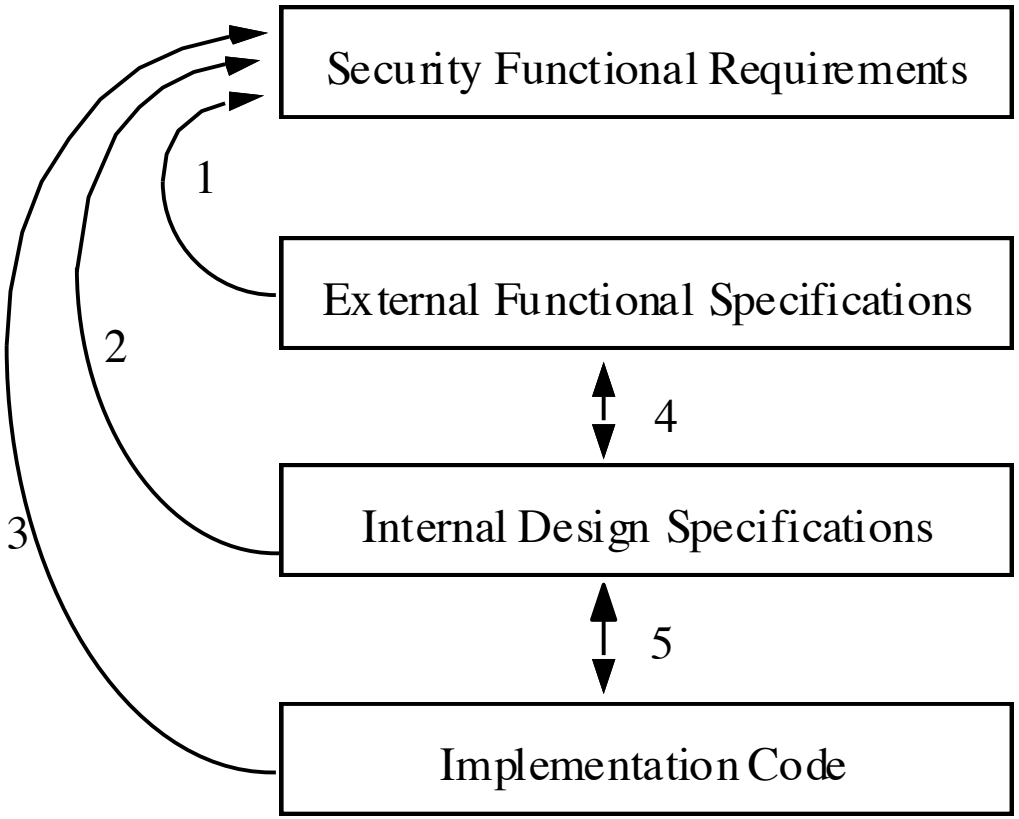
Formal Specifications

- 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