Waterfall Life Cycle Model

- Requirements definition and analysis
  - Functional and non-functional
  - General (for customer), specifications
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Relationship of Stages
Models

- Exploratory programming
  - Develop working system quickly
  - Used when detailed requirements specification cannot be formulated in advance, and adequacy is goal
  - No requirements or design specification, so low assurance
- Prototyping
  - Objective is to establish system requirements
  - Future iterations (after first) allow assurance techniques

Models

- Formal transformation
  - Create formal specification
  - Translate it into program using correctness-preserving transformations
  - Very conducive to assurance methods
- System assembly from reusable components
  - Depends on whether components are trusted
  - Must assure connections, composition as well
  - Very complex, difficult to assure
Models

- Extreme programming
  - Rapid prototyping and “best practices”
  - Project driven by business decisions
  - Requirements open until project complete
  - Programmers work in teams
  - Components tested, integrated several times a day
  - Objective is to get system into production as quickly as possible, then enhance it
  - Evidence adduced after development needed for assurance

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
Auditing

- Overview
- What is auditing?
- What does an audit system look like?
- How do you design an auditing system?
- Auditing mechanisms
- Examples: NFSv2, LAFS

What is Auditing?

- Logging
  - Recording events or statistics to provide information about system use and performance
- Auditing
  - Analysis of log records to present information about the system in a clear, understandable manner
Uses

- Describe security state
  - Determine if system enters unauthorized state
- Evaluate effectiveness of protection mechanisms
  - Determine which mechanisms are appropriate and working
  - Deter attacks because of presence of record

Problems

- What do you log?
  - Hint: looking for violations of a policy, so record at least what will show such violations
- What do you audit?
  - Need not audit everything
  - Key: what is the policy involved?
Audit System Structure

- Logger
  - Records information, usually controlled by parameters
- Analyzer
  - Analyzes logged information looking for something
- Notifier
  - Reports results of analysis

Logger

- Type, quantity of information recorded controlled by system or program configuration parameters
- May be human readable or not
  - If not, usually viewing tools supplied
  - Space available, portability influence storage format
Example: RACF

- Security enhancement package for IBM’s MVS/VM
- Logs failed access attempts, use of privilege to change security levels, and (if desired) RACF interactions
- View events with LISTUSERS commands

RACF: Sample Entry

USER=EW125004  NAME=S.J.TURNER  OWNER=SECADM  CREATED=88.004
DEFAULT-GROUP=HUMRES  PASSDATE=88.004  PASS-INTERVAL=30
ATTRIBUTES=ADSP
REVOKE DATE=NONE  RESUME-DATE=NONE
LAST-ACCESS=88.020/14:15:10
CLASS AUTHORIZATIONS=NONE
NO-INSTALLATION-DATA
NO-MODEL-NAME
LOGON ALLOWED (DAYS) (TIME)

ANYDAY ANYTIME
GROUP=HUMRES AUTH=JOIN CONNECT-OWNER=SECADM
CONNECT-DATE=88.004
CONNECTS=15 UAACC=READ LAST-CONNECT=88.018/16:45:06
CONNECT ATTRIBUTES=NONE
REVOKE DATE=NONE RESUME DATE=NONE
GROUP=PERSNL AUTH=JOIN CONNECT-OWNER=SECADM CONNECT-DATE=88.004
CONNECTS=25 UAACC=READ LAST-CONNECT=88.020/14:15:10
CONNECT ATTRIBUTES=NONE
REVOKE DATE=NONE RESUME DATE=NONE
SECURITY-LEVEL=NONE SPECIFIED
CATEGORY AUTHORIZATION=NONE SPECIFIED
Example: Windows NT

- Different logs for different types of events
  - *System event* logs record system crashes, component failures, and other system events
  - *Application event* logs record events that applications request be recorded
  - *Security event* log records security-critical events such as logging in and out, system file accesses, and other events
- Logs are binary; use *event viewer* to see them
- If log full, can have system shut down, logging disabled, or logs overwritten

Windows NT Sample Entry

- **Date:** 2/12/2000
- **Source:** Security
- **Time:** 13:03
- **Category:** Detailed Tracking
- **Type:** Success
- **EventID:** 592
- **User:** WINDSOR\Administrator
- **Computer:** WINDSOR
- **Description:**
  A new process has been created:
  - **New Process ID:** 2216594592
  - **Image File Name:** Program Files\Internet Explorer\IEXPLORE.EXE
  - **Creator Process ID:** 2217918496
  - **User Name:** Administrator
  - **FDomain:** WINDSOR
  - **Logon ID:** (0x0x0x14B4c4)

[would be in graphical format]
Analyzer

• Analyzes one or more logs
  – Logs may come from multiple systems, or a single system
  – May lead to changes in logging
  – May lead to a report of an event

Examples

• Using `swatch` to find instances of `telnet` from `tcpd` logs:
  `/telnet/&!/localhost/&!/*.site.com/`
• Query set overlap control in databases
  – If too much overlap between current query and past queries, do not answer
• Intrusion detection analysis engine (director)
  – Takes data from sensors and determines if an intrusion is occurring
Notifier

• Informs analyst, other entities of results of analysis
• May reconfigure logging and/or analysis on basis of results

Examples

• Using `swatch` to notify of `telnets`
  
  /telnet/&!/localhost/&!/*.site.com/ mail staff

• Query set overlap control in databases
  – Prevents response from being given if too much overlap occurs

• Three failed logins in a row disable user account
  – Notifier disables account, notifies sysadmin
Designing an Audit System

- Essential component of security mechanisms
- Goals determine what is logged
  - Idea: auditors want to detect violations of policy, which provides a set of constraints that the set of possible actions must satisfy
  - So, audit functions that may violate the constraints
- Constraint $p_i: action \Rightarrow condition$

Example: Bell-LaPadula

- Simple security condition and *-property
  - $S$ reads $O \Rightarrow L(S) \geq L(O)$
  - $S$ writes $O \Rightarrow L(S) \leq L(O)$
  - To check for violations, on each read and write, must log $L(S), L(O)$, action (read, write), and result (success, failure)
  - Note: need not record $S, O$!
    - In practice, done to identify the object of the (attempted) violation and the user attempting the violation
Remove Tranquility

- New commands to manipulate security level must also record information
  - $S$ reclassify $O$ to $L(O') \Rightarrow L(O) \leq L(S)$ and $L(O') \leq L(S)$
  - Log $L(O)$, $L(O')$, $L(S)$, action (reclassify), and result (success, failure)
  - Again, need not record $O$ or $S$ to detect violation
    - But needed to follow up …

Example: Chinese Wall

- Subject $S$ has $COI(S)$ and $CD(S)$
  - $CD_H(S)$ is set of company datasets that $S$ has accessed
- Object $O$ has $COI(O)$ and $CD(O)$
  - $san(O)$ iff $O$ contains only sanitized information
- Constraints
  - $S$ reads $O \Rightarrow COI(O) \neq COI(S) \lor \exists \hat{O}(CD(\hat{O}) \in CD_H(S))$
  - $S$ writes $O \Rightarrow (S$ canread $O) \land \neg \exists \hat{O}(COI(O) = COI(\hat{O}) \land S$ canread $\hat{O} \land \neg san(\hat{O}))$
Recording

- \( S \) reads \( O \Rightarrow COI(O) \neq COI(S) \lor \exists \hat{O}(CD(\hat{O}) \in CD_{\hat{O}}(S)) \)
  - Record \( COI(O), COI(S), CD_{\hat{O}}(S), CD(\hat{O}) \) if such an \( \hat{O} \) exists, action (read), and result (success, failure)
- \( S \) writes \( O \Rightarrow (S \text{ can read } O) \land \neg \exists \hat{O}(COI(O) = COI(\hat{O}) \land S \text{ can read } \hat{O} \land \neg san(\hat{O})) \)
  - Record \( COI(O), COI(S), CD_{\hat{O}}(S), \) plus \( COI(\hat{O}) \) and \( CD(\hat{O}) \) if such an \( \hat{O} \) exists, action (write), and result (success, failure)

Implementation Issues

- Show non-security or find violations?
  - Former requires logging initial state as well as changes
- Defining violations
  - Does “write” include “append” and “create directory”?
- Multiple names for one object
  - Logging goes by object and not name
  - Representations can affect this (if you read raw disks, you’re reading files; can your auditing system determine which file?)
Syntactic Issues

• Data that is logged may be ambiguous
  – BSM: two optional text fields followed by two mandatory text fields
  – If three fields, which of the optional fields is omitted?
• Solution: use grammar to ensure well-defined syntax of log files

Example

```plaintext
entry : date host prog [ bad ] user [ "from" host ] "to" user "on" tty

date : daytime
host : string
prog : string ":/"
bad : "FAILED"
user : string
tty : ":/dev/" string
```

• Log file entry format defined unambiguously
• Audit mechanism could scan, interpret entries without confusion
More Syntactic Issues

- Context
  - Unknown user uses anonymous `ftp` to retrieve file “/etc/passwd”
  - Logged as such
  - Problem: which `/etc/passwd` file?
    - One in system `/etc` directory
    - One in anonymous `ftp` directory `/var/ftp/etc`, and as `ftp` thinks `/var/ftp` is the root directory, `/etc/passwd` refers to `/var/ftp/etc/passwd`

Log Sanitization

- $U$ set of users, $P$ policy defining set of information $C(U)$ that $U$ cannot see; log sanitized when all information in $C(U)$ deleted from log
- Two types of $P$
  - $C(U)$ can’t leave site
    - People inside site are trusted and information not sensitive to them
  - $C(U)$ can’t leave system
    - People inside site not trusted or (more commonly) information sensitive to them
    - Don’t log this sensitive information
Logging Organization

- Top prevents information from leaving site
  - Users’ privacy not protected from system administrators, other administrative personnel
- Bottom prevents information from leaving system
  - Data simply not recorded, or data scrambled before recording

Reconstruction

- Anonymizing sanitizer cannot be undone
  - No way to recover data from this
- Pseudonymizing sanitizer can be undone
  - Original log can be reconstructed
- Importance
  - Suppose security analysis requires access to information that was sanitized?
**Issue**

- Key: sanitization must preserve properties needed for security analysis
- If new properties added (because analysis changes), may have to resanitize information
  - This *requires* pseudonymous sanitization or the original log

**Example**

- Company wants to keep its IP addresses secret, but wants a consultant to analyze logs for an address scanning attack
  - Connections to port 25 on IP addresses 10.163.5.10, 10.163.5.11, 10.163.5.12, 10.163.5.13, 10.163.5.14, 10.163.5.15
  - Sanitize with random IP addresses
    - Cannot see sweep through consecutive IP addresses
  - Sanitize with sequential IP addresses
    - Can see sweep through consecutive IP addresses
Generation of Pseudonyms

1. Devise set of pseudonyms to replace sensitive information
   • Replace data with pseudonyms
   • Maintain table mapping pseudonyms to data

2. Use random key to encipher sensitive datum and use secret sharing scheme to share key
   • Used when insiders cannot see unsanitized data, but outsiders (law enforcement) need to
   • Requires $t$ out of $n$ people to read data

Application Logging

• Applications logs made by applications
  – Applications control what is logged
  – Typically use high-level abstractions such as:
    su: bishop to root on /dev/tty0
  – Does not include detailed, system call level information such as results, parameters, etc.
System Logging

- Log system events such as kernel actions
  - Typically use low-level events
    - Does not include high-level abstractions such as loading libraries (as above)

Contrast

- Differ in focus
  - Application logging focuses on application events, like failure to supply proper password, and the broad operation (what was the reason for the access attempt?)
  - System logging focuses on system events, like memory mapping or file accesses, and the underlying causes (why did access fail?)
- System logs usually much bigger than application logs
- Can do both, try to correlate them
Design

• *A posteriori* design
  – Need to design auditing mechanism for system not built with security in mind

• Goal of auditing
  – Detect *any* violation of a stated policy
    • Focus is on policy and actions designed to violate policy; specific actions may not be known
  – Detect actions *known* to be part of an attempt to breach security
    • Focus on specific actions that have been determined to indicate attacks

Detect Violations of Known Policy

• Goal: does system enter a disallowed state?
• Two forms
  – State-based auditing
    • Look at current state of system
  – Transition-based auditing
    • Look at actions that transition system from one state to another
State-Based Auditing

• Log information about state and determine if state allowed
  – Assumption: you can get a snapshot of system state
  – Snapshot needs to be consistent
  – Non-distributed system needs to be quiescent
  – Distributed system can use Chandy-Lamport algorithm, or some other algorithm, to obtain this

Example

• File system auditing tools
  – Thought of as analyzing single state (snapshot)
  – In reality, analyze many slices of different state unless file system quiescent
  – Potential problem: if test at end depends on result of test at beginning, relevant parts of system state may have changed between the first test and the last
    • Classic TOCTTOU flaw
Transition-Based Auditing

- Log information about action, and examine current state and proposed transition to determine if new state would be disallowed
  - Note: just analyzing the transition may not be enough; you may need the initial state
  - Tend to use this when specific transitions *always* require analysis (for example, change of privilege)

Example

- TCP access control mechanism intercepts TCP connections and checks against a list of connections to be blocked
  - Obtains IP address of source of connection
  - Logs IP address, port, and result (allowed/blocke) in log file
  - Purely transition-based (current state not analyzed at all)