

IS 2150 / TEL 2810

Information Security & Privacy



James Joshi
Associate Professor, SIS

Lecture 9
Oct 30, 2013

Authentication, Identity
Vulnerability Analysis



Objectives

- Understand/explain the issues related to, and utilize the techniques
 - Authentication and identification
 - Vulnerability analysis/classification
 - Techniques
 - Taxonomy



Authentication and Identity



What is Authentication?

- Authentication:
 - Binding identity and external entity to subject
- How do we do it?
 - Entity *knows* something (secret)
 - Passwords, id numbers
 - Entity *has* something
 - Badge, smart card
 - Entity *is* something
 - Biometrics: fingerprints or retinal characteristics
 - Entity is in *someplace*
 - Source IP, restricted area terminal

Authentication System: Definition



- A : Set of *authentication information*
 - used by entities to prove their identities (e.g., password)
- C : Set of *complementary information*
 - used by system to validate authentication information (e.g., hash of a password or the password itself)
- F : Set of *complementation functions* (to generate C)
 - $f: A \rightarrow C$
 - Generate appropriate $c \in C$ given $a \in A$
- L : set of *authentication functions*
 - $l: A \times C \rightarrow \{ \mathbf{true}, \mathbf{false} \}$
 - verify identity
- S : set of *selection functions*
 - Generate/alter A and C
 - e.g., commands to change password

Authentication System: Passwords

- Example: plaintext passwords

- $A = C = \text{alphabet}^*$
- f returns argument: $f(a)$ returns a
- $/$ is string equivalence: $\ell(a, b)$ is true if $a = b$

- Complementation Function

- Null (return the argument as above)
 - requires that c be protected; i.e. password file needs to be protected
- One-way hash – function such that
 - *Complementary information* $c = f(a)$ easy to compute
 - $f^{-1}(c)$ difficult to compute





Passwords

- Example: Original Unix
 - A password is up to eight characters
 - each character could be one of 127 possible characters;
 - *A* contains approx. 6.9×10^{16} passwords
 - Password is hashed using one of 4096 functions into a 11 character string
 - 2 characters pre-pended to indicate the hash function used
 - *C* contains passwords of size 13 characters, each character from an alphabet of 64 characters
 - Approximately 3.0×10^{23} strings
 - Stored in file */etc/passwd* (all can read)



Authentication System

- Goal: identify the entities correctly
- Approaches to protecting
 - Hide enough information so that one of a , c or f cannot be found
 - Make C readable only to root
 - Make F unknown
 - Prevent access to the authentication functions L
 - *root* cannot log in over the network



Attacks on Passwords

- Dictionary attack: Trial and error guessing
 - Type 1: attacker knows A, F, C
 - Guess g and compute $f(g)$ for each f in F
 - Type 2: attacker knows A, I
 - I returns **True** for guess g
- Counter: Difficulty based on $|A|$, Time
 - Probability P of breaking a password
 - G be the number of guesses that can be tested in one time unit
 - $|A| \geq TG/P$
 - Assumptions:
 - time constant; all passwords are equally likely



Password Selection

- Random
 - Depends on the quality of random number generator;
 - Size of legal passwords
 - 8 characters: humans can remember only one
- Pronounceable nonsense
 - Based on unit of sound (phoneme)
 - Easier to remember
- User selection (proactive selection)
 - Controls on allowable
 - At least 1 digit, 1 letter, 1 punctuation, 1 control character
 - Obscure poem verse



Password Selection

- Reusable Passwords susceptible to dictionary attack (type 1)
 - *Salting* can be used to increase effort needed
 - makes the choice of complementation function a function of randomly selected data
 - Random data is different for different user
 - Authentication function is chosen on the basis of the salt
 - Many Unix systems:
 - A salt is randomly chosen from 0..4095
 - Complementation function depends on the salt



Password Selection

- Password aging
 - Change password after some time: based on expected time to guess a password
 - Disallow change to previous n passwords
- Fundamental problem is *reusability*
 - Replay attack is easy
 - Solution:
 - Authenticate in such a way that the transmitted password changes each time



Authentication Systems: Challenge-Response

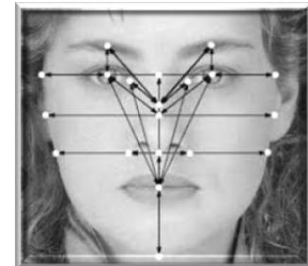
- Pass algorithm
 - authenticator sends message m
 - subject responds with $f(m)$
 - f is a secret encryption function
 - Example: ask for second input based on some algorithm

Authentication Systems: Challenge-Response

- One-time password: *invalidated after use*
 - f changes after use
- S/Key uses a hash function (MD4/MD5)
 - User chooses an initial seed k
 - Key generator calculates
 - $k_1 = h(k), k_2 = h(k_1) \dots, k_n = h(k_{n-1})$
 - Passwords used in the order
 - $p_1 = k_n, p_2 = k_{n-1}, \dots, p_n = k_1$
 - Suppose $p_1 = k_n$ is intercepted;
 - the next password is $p_2 = k_{n-1}$
 - Since $h(k_{n-1}) = k_n$ the attacker needs to invert h to determine the next password

Authentication Systems: Biometrics

- Fingerprint (optical scanning)
 - Camera's needed (bulky)
- Voice
 - Speaker-verification (identity) or speaker-recognition (info content)
- Iris/retina patterns (unique for each person)
 - Laser beaming is intrusive
- Face recognition
 - Facial features can make this difficult
- Keystroke interval/timing/pressure



Authentication Systems

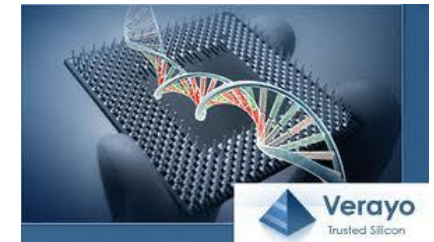
- CAPTCHA

- Completely Automated Public Turing Test To Tell Computers and Humans Apart



- Physically Unclonable Functions

- E.g., RFID counterfeit detection





Attacks on Biometrics

- Fake biometrics
 - fingerprint “mask”
 - copy keystroke pattern
- Fake the interaction between device and system
 - Replay attack
 - Requires careful design of entire authentication system



Vulnerability Analysis



Vulnerability Analysis

- **Vulnerability or security flaw:** specific failures of security controls (procedures, technology or management)
 - Errors in code
 - Human violators
 - Mismatch between assumptions
- **Exploit:** Use of vulnerability to violate policy
- **Attacker:** Attempts to exploit the vulnerability



Techniques for Detecting Vulnerabilities

- System Verification
 - Determine preconditions, post-conditions
 - Validate that system ensures post-conditions given preconditions
 - Can** prove the absence of vulnerabilities
- Penetration testing
 - Start with system/environment characteristics
 - Try to find vulnerabilities
 - Can not** prove the absence of vulnerabilities



Types/layers of Penetration Testing

- Black Box (External Attacker)
 - External attacker has no knowledge of target system
 - Attacks built on human element – Social Engineering
- System access provided (External Attacker)
 - 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

Flaw Hypothesis Methodology:

- Information gathering
 - Examine design, environment, system functionality
- Flaw hypothesis
 - Predict likely vulnerabilities
- Flaw testing
 - Determine where vulnerabilities exist
- Flaw generalization
 - Attempt to broaden discovered flaws
- Flaw elimination (often not included)
 - Suggest means to eliminate flaw

Flaw does
Not exist

Refine with new
understanding



Problems with Penetration Testing

- Nonrigorous
 - Dependent on insight (and whim) of testers
 - No good way of evaluating when “complete”
- How do we make it systematic?
 - Try all classes of likely flaws
 - *But what are these?*
- Vulnerability Classification!



Vulnerability Classification

- Goal: describe spectrum of possible flaws
 - Enables design to avoid flaws
 - Improves coverage of penetration testing
 - Helps design/develop intrusion detection
- How do we classify?
 - By how they are exploited?
 - By where they are found?
 - By the nature of the vulnerability?



Example flaw: `xterm` log

- *xterm* runs as root
 - Generates a log file
 - Appends to log file if file exists
- Problem: In `/etc/passwd` log_file
- Solution

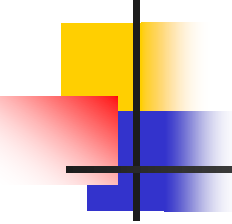
```
if (access("log_file", W_OK) == 0)
    If ((fd = open("log_file", O_WRONLY|O_APPEND)) < 0) {
        - error handling
    }
```

What can go wrong?



Example: Finger Daemon *(exploited by Morris worm)*

- *finger* sends name to *fingerd*
 - *fingerd* allocates 512 byte buffer on stack
 - Places name in buffer
 - Retrieves information (local finger) and returns
- Problem: If name > 512 bytes, overwrites return address
- Exploit: Put code in "name", pointer to code in bytes 513+
 - Overwrites return address



RISOS: Research Into Secure Operating Systems (7 Classes)

1. Incomplete parameter validation
 - E.g., buffer overflow –
2. Inconsistent parameter validation
 - Different routines with different formats for same data
3. Implicit sharing of privileged / confidential data
 - OS fails to isolate processes and users
4. Asynchronous validation / inadequate serialization
 - Race conditions and TOCTTOU flaws
5. Inadequate identification / authentication / authorization
 - Trojan horse; accounts without passwords
6. Violable prohibition / limit
 - Improper handling of bounds conditions (e.g., in memory allocation)
7. Exploitable logic error
 - Incorrect error handling, incorrect resource allocations etc.



Protection Analysis Model Classes

- 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

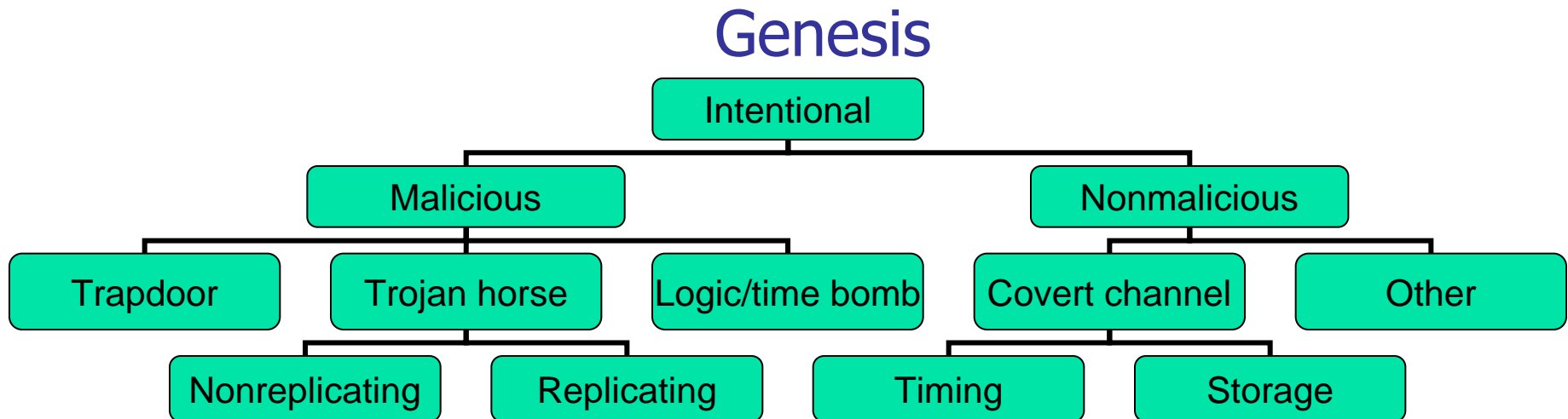


PA flaw classes

1. Improper protection domain initialization and enforcement
 - a. *domain*: Improper choice of initial protection domain
 - b. *exposed representations*: Improper isolation of implementation detail (Covert channels)
 - c. *consistency of data over time*: Improper change
 - d. *naming*: Improper naming (two objects with same name)
 - e. *residuals*: Improper deallocation or deletion
2. Improper validation *validation of operands, queue management dependencies*:
3. Improper synchronization
 - a. *interrupted atomic operations*: Improper indivisibility
 - b. *serialization*: Improper sequencing
4. Improper choice of operand or operation *critical operator selection errors*

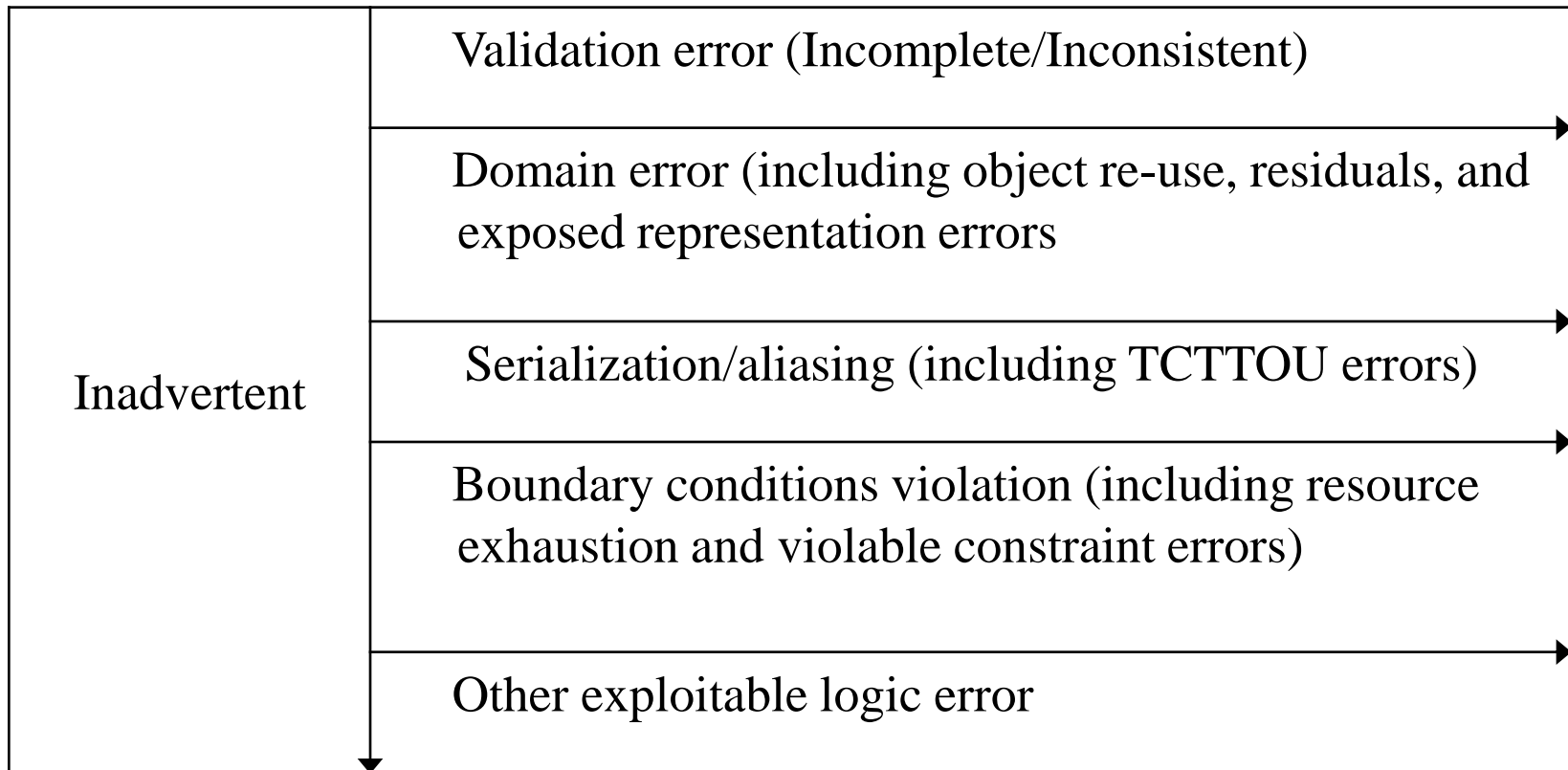
NRL Taxonomy

- Three classification schemes
 - How did it enter
 - When was it “created”
 - Where is it

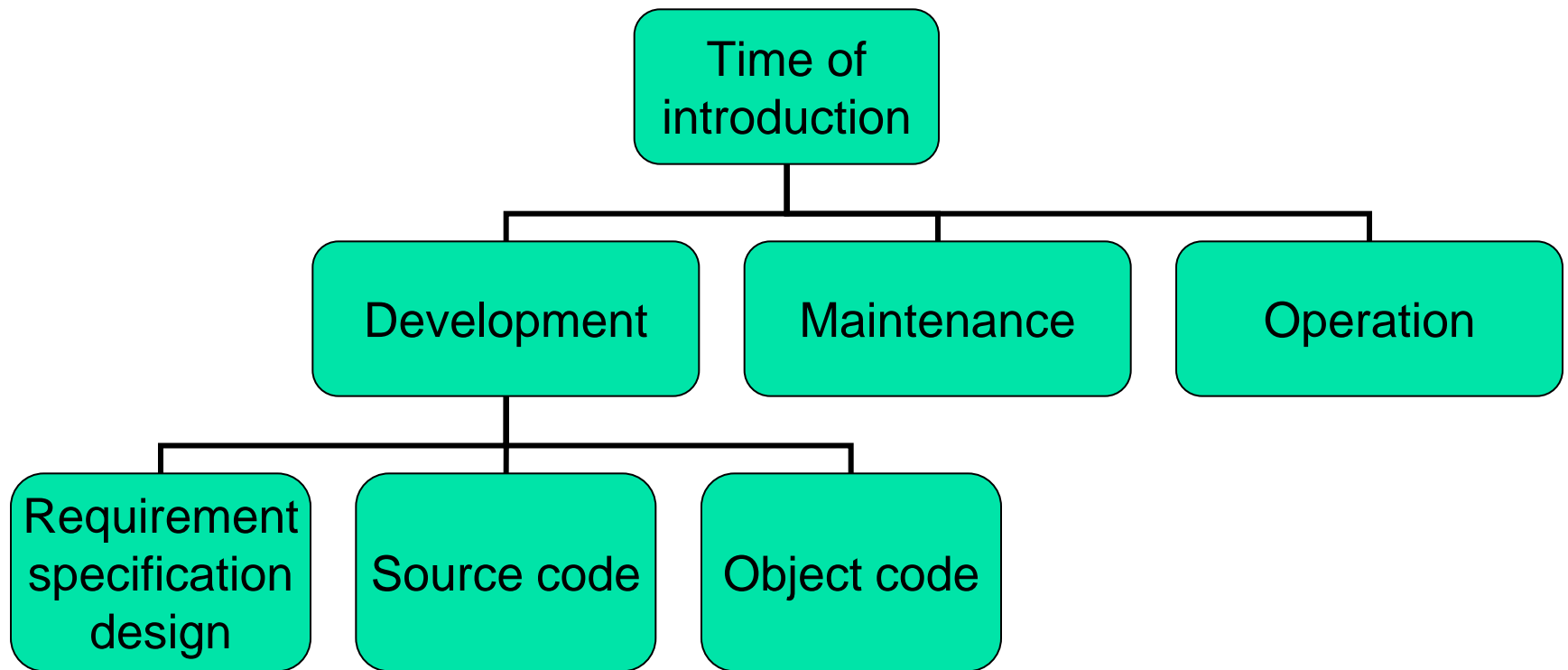




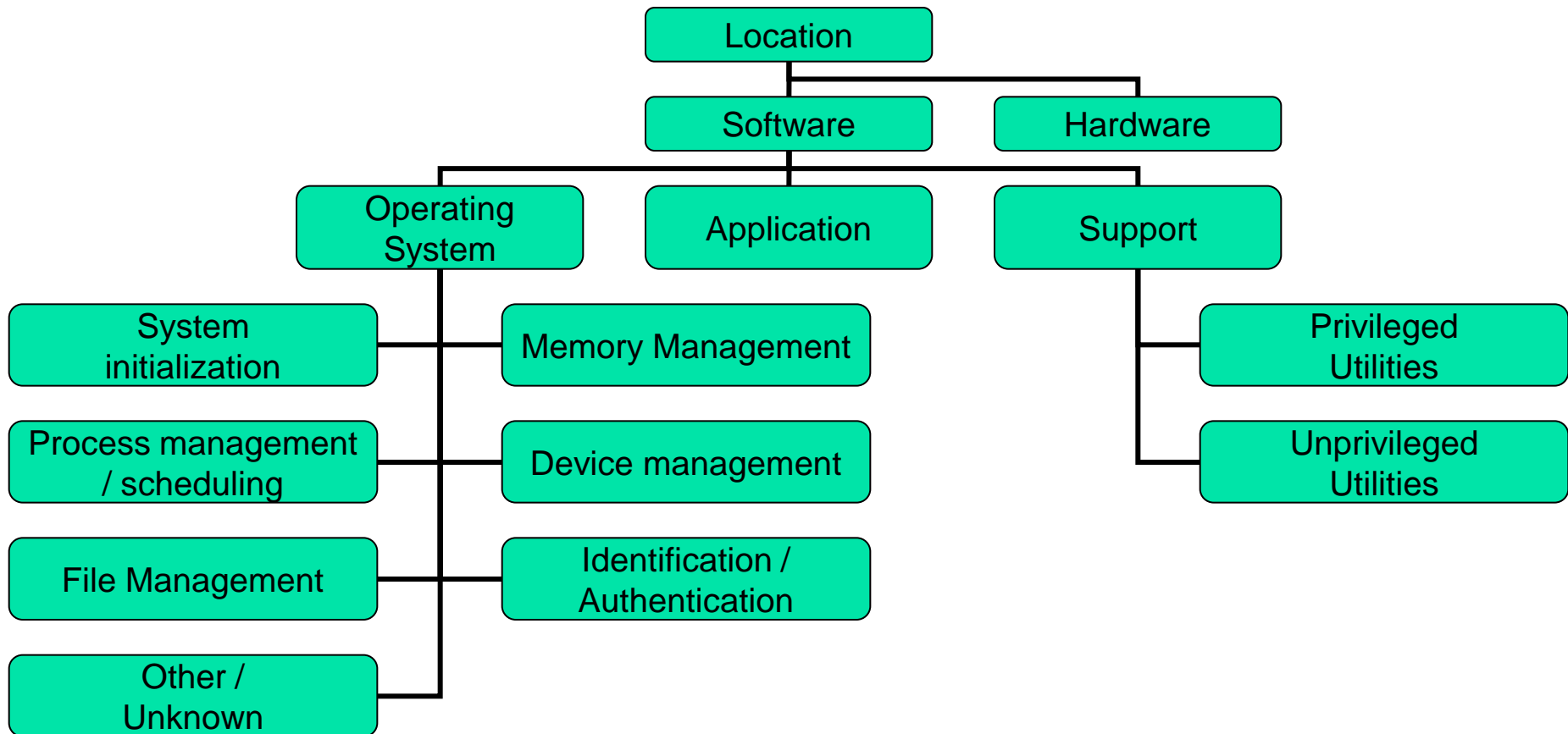
NRL Taxonomy (Genesis)



NRL Taxonomy: Time



NRL Taxonomy: Location





Aslam's Model

- Attempts to classify faults unambiguously
 - Decision procedure to classify faults
- Coding Faults
 - Synchronization errors
 - Timing window
 - Improper serialization
 - Condition validation errors
 - Bounds not checked
 - Access rights ignored
 - Input not validated
 - Authentication / Identification failure
- Emergent Faults
 - Configuration errors
 - Wrong install location
 - Wrong configuration information
 - Wrong permissions
 - Environment Faults

Common Vulnerabilities and Exposures (cve.mitre.org)

- Captures *specific* vulnerabilities
 - Standard name
 - Cross-reference to CERT, etc.
- Entry has three parts
 - Unique ID
 - Description
 - References

Name	CVE-1999-0965
Description	Race condition in xterm allows local users to modify arbitrary files via the logging option.

References

- CERT:CA-93.17
- XF:xterm