BASIC CONCEPTS AND TAXONOMY OF DEPENDABLE AND SECURE COMPUTING

by Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr,
IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 1, NO. 1, JANUARY-MARCH 2004

BASICS

• We have discussed the basic issues of dependable systems before.

• Now we will focus more on survivability-related issues of the aforementioned paper

• Most of the material is directly taken from the paper and (to avoid visual clutter) will not be explicitly cited!
2) BASIC CONCEPTS

• **System**
  • *entity* that interacts with other *entities*
  • includes hardware, software, humans, physical world with its natural phenomena

• **system boundary**
  • *function* is what it should do, often is described by functional specification in terms of functionality and performance
  • *behavior* is what system does to implement its functions
  • behavior is described by sequence of *states*

• **Total State** of a System defined by following states:
  • computation
  • communication
  • stored information
  • interconnection
  • physical condition
2) BASIC CONCEPTS

• **Structure of a system**
  • set of **components** that interact
  • each component is another system
    • recursive definition
    • stops with atomic component
      • i.e., no need or not possible to further break down

• **Service** delivered by a system
  • in its role as **provider**
  • **user** is another system receiving service from the provider
  • **service interface** is the boundary where service delivery takes place
    • user sees **external state** of provider; remaining part is **internal state**
  • user receives service at **use interface**
2) BASIC CONCEPTS

• Threats to Dependability and Security
  • **Service failure**, or just **failure**
    • delivered service deviates from correct service
  • **transition** from correct to incorrect service

2) BASIC CONCEPTS

• Threats to Dependability and Security
  • **Service outage**
    • period of delivery of incorrect service
  • **Service restoration**
    • transition from incorrect to correct service
    • deviation from correct service may assume different forms: **service failure modes**
2) BASIC CONCEPTS

- Failure, error, fault
  - Service is sequence of system’s external states
  - Service failure means $\exists$ at least one external state of the system deviates from the correct service state
  - That deviation is called an **error**
  - The cause of the error is called **fault**

2) BASIC CONCEPTS

- Faults
  - internal fault or external
  - **vulnerability**, i.e., an internal fault that enables an external fault to harm the system, is necessary for an external fault to cause an error and possibly subsequent failure
2) BASIC CONCEPTS

• typically: **fault** causes **error**, which can cause **failure**
  • fault is **active** when it causes an error
  • otherwise it is **dormant**

2) BASIC CONCEPTS

• If functional specification of a system includes a set of several functions, then
  • failure of one or more services that implement the function may leave system in **degraded mode**
    • still offers subset of needed services
    • e.g., slower, limited service, emergency service
  • system is said to have suffered **partial failure**
2) BASIC CONCEPTS

• Dependability Security and their Attributes
  • original definition of **dependability**
    • “ability to deliver service that can justifiably be trusted”
  • alternate definition
    • “ability to avoid service failures that are more frequent and more severe than is acceptable”

• Trust
  • dependence of system A on system B represents the extend to which system A’s dependability is affected by that of system B
  • concept of dependence leads to that of **trust**, 
    • trust = accepted dependence
2) BASIC CONCEPTS

• Dependability encompasses the following attributes
  • **availability**: readiness for correct service.
  • **reliability**: continuity of correct service.
  • **safety**: absence of catastrophic consequences on the user(s) and the environment.
  • **integrity**: absence of improper system alterations.
  • **maintainability**: ability to undergo modifications and repairs.

2) BASIC CONCEPTS

• when addressing security we add
  • **confidentiality**, the absence of unauthorized disclosure of information
  • Security is composite of the attributes
    • confidentiality
    • integrity
    • availability
2) BASIC CONCEPTS

- Dependability and security attributes

[Diagram]

Dependability → Availability, Reliability, Safety, Confidentiality, Integrity, Maintainability → Security

2) BASIC CONCEPTS

- Dependability and security tree

[Diagram]

2) BASIC CONCEPTS

• Means to attain dependability and security:
  • **Fault prevention**: prevent the occurrence or introduction of faults.
  • **Fault tolerance**: avoid service failures in the presence of faults.
  • **Fault removal**: reduce the number and severity of faults.
  • **Fault forecasting**: estimate the present number, the future incidence, and the likely consequences of faults.

3) THREATS TO DEPENDABILITY AND SECURITY

• 3.1: System Life Cycle: Phases and Environment
  • Development phase: all activities from initial concept to green light
    • Development Environment of system consists of
      • **physical world** with its natural phenomena
      • **human developers** (+lacking competence, malicious objective)
      • **development tools**: software and hardware
      • **production and test facilities**
3) THREATS TO DEPENDABILITY AND SECURITY

• Use phase
  • System is accepted for use and starts delivering services.
  • Alternating periods of:
    Service delivery
    Service outage
    Service shutdown

• Maintenance may take place during all three periods of use phase

USE ENVIRONMENT ELEMENTS:

• **Physical world**: with its natural phenomena

• **Administrators** (includes maintainers): have authority to manage, modify, repair and use system. Some authorized humans may lack competence of have malicious objectives
USE ENVIRONMENT ELEMENTS:

- **Users**: humans or other system that receive services

- **Providers**: humans or other systems that deliver services

- **Infrastructure**: entities that provide services to the system, e.g., information sources (time, GPS) communications equipment/links, power, cooling etc.

USE ENVIRONMENT ELEMENTS:

- **Intruders**: malicious entities (human or other systems)
  - attempt to exceed authority they have
  - alter services
  - halt them
  - alter system’s functionality or performance
  - access confidential information
  - examples: hackers, vandals, corrupt insiders, governments, malicious software
MAINTENANCE

Fig. 3. The various forms of maintenance.

FAULTS: OVERVIEW

Fig. 4. The elementary fault classes.

Development faults
[occur during (a) system development, (b) maintenance during the use phase, and (c) generation of procedures to operate or to maintain the system]

Operational faults
[occur during service-delivery of the use phase]

Internal faults
[originate inside the system boundary]

External faults
[originate outside the system boundary and propagate errors into the system by interaction or interference]

Natural faults
[caused by natural phenomena without human participation]

Human-Made faults
[result from human actions]

Hardware faults
[originate in, or affect, hardware]

Software faults
[affect software, i.e., programs or data]

Malicious faults
[introduced by a human with the malicious objective of causing harm to the system]

Non-Malicious faults
[introduced without a malicious objective]

Deliberate faults
[result of a harmful decision]

Non-Deliberate faults
[introduced without awareness]

Accidental faults
[introduced inadvertently]

Incompetence faults
[result from lack of professional competence by the authorized human(s), or from inadequacy of the development organization]

Permanent faults
[presence is assumed to be continuous in time]

Transient faults
[presence is bounded in time]
The two basic classes of human-made faults are distinguished by the objective of the developer or of the humans interacting with the system during its use:

- **Malicious faults**, introduced during either system development with the objective to cause harm to the system during its use (5-6), or directly during use (22-25).

- **Nonmalicious faults** (1-4, 7-21, 26-31), introduced without malicious objectives.

We consider nonmalicious faults first. They can be partitioned according to the developer's intent:

- **Nondeliberate faults** that are due to mistakes, that is, unintended actions of which the developer, operator, maintainer, etc. is not aware (1, 2, 7, 8, 16-18, 26-28);

- **Deliberate faults** that are due to bad decisions, that is, intended actions that are wrong and cause faults (3, 4, 9, 10, 19-21, 29-31).

Deliberate, nonmalicious, development faults (3, 4, 9, 10) result generally from trade offs, either 1) aimed at preserving acceptable performance, at facilitating system utilization, or 2) induced by economic considerations.

Deliberate, nonmalicious interaction faults (19-21, 29-31) may result from the action of an operator either aimed at overcoming an unforeseen situation, or deliberately violate...
3.2.3 ON HUMAN-MADE FAULTS

• Non-malicious faults
  • introduced without malicious objectives
  • **non-deliberate fault**: due to mistakes, i.e., unintended action, developer/operator/maintainer is not aware
  • **deliberate fault**: due to bad decisions, i.e., unintended action that are wrong and cause faults
### NON-MALICIOUS FAULTS

- Incompetence faults
  - individual, group, organization
  - e.g., Advance Automation System to replace aging USA air traffic control system
NON-MALICIOUS FAULTS

• Deployment faults
  • hardware
    • e.g., HW “errata” are listed in specification updates
    • may continue during lifetime of the product
  • software
    • software aging: progressively accrued error conditions cause performance degradation of failure
    • e.g., memory bloating/leaking, unterminated threads, storage space fragmentation, accumulation of round-off errors, ...

3.2.4 ON MALICIOUS FAULTS

• Malicious human-made faults
  • typical goals:
    • disrupt or halt service => denial of service
    • access confidential information
    • improperly modify the systems
3.2.4 ON MALICIOUS FAULTS

- Malicious logic faults
  - development faults: e.g., Trojan horses, logic or timing bombs, trapdoors
  - operational faults: e.g. viruses, worms, zombies
- Intrusion attempts
  - operational external faults. May be performed by system operators/admins
  - may use physical means to cause faults, e.g., power fluctuation, radiation, wire-tapping, heating/cooling

**logic bomb**: malicious logic that remains dormant in the host system till a certain time or an event occurs, or certain conditions are met, and then deletes files, slows down or crashes the host system, etc.

**Trojan horse**: malicious logic performing, or able to perform, an illegitimate action while giving the impression of being legitimate; the illegitimate action can be the disclosure or modification of information (attack against confidentiality or integrity) or a logic bomb;

**trapdoor**: malicious logic that provides a means of circumventing access control mechanisms;

**virus**: malicious logic that replicates itself and joins another program when it is executed, thereby turning into a Trojan horse; a virus can carry a logic bomb;

**worm**: malicious logic that replicates itself and propagates without the users being aware of it; a worm can also carry a logic bomb;

**zombie**: malicious logic that can be triggered by an attacker in order to mount a coordinated attack.
3.2.5 ON INTERACTION FAULTS

- Occur in use phase
  - elements of the use environment interaction with the system
  - all external
  - human-made
- Examples
  - configuration faults, reconfiguration faults

3.3 FAILURES

- Service failure
  - def.: event that occurs when the delivered service deviates from correct service
  - service failure modes: different ways in which deviation is manifested
  - content failure: content of info delivered deviates from implementing the system function
  - timing failure: time of arrival (early or late) or duration of info delivered at service interface deviates from implementing the system function.
3.3 FAILURES

• Service failure cont.
  • both information and timing are incorrect:
    • halt failure: external state becomes constant
    • silent failure: no service is delivered at interface
    • erratic failure: service is delivered (not halted) but is erratic, e.g. babbling

Fig. 8. Service failure modes with respect to the failure domain viewpoint.
3.3 FAILURES

• Consistency

• consistent failures: incorrect service is perceived identically by all system users

• inconsistent failures: some of all users perceive differently incorrect service. Byzantine failures

SERVICE FAILURE MODES

Fig. 9. Service failure modes.
3.3.2 DEVELOPMENT FAILURES

- Budget failure
  - “broke” before system passes acceptance testing
- Schedule failure
  - schedule slips to a point in the future where the system would be technologically obsolete or functionally inadequate for user’s needs

3.5 FAULTS, ERRORS AND FAILURES

![Error Propagation Diagram](attachment:Fig_10.png)

Fig. 10 Error Propagation
EXAMPLES

• traditional hardware fault tolerance view
  • physical fault (may be dormant), e.g., stuck-at
  • produces error
  • may result in failure

EXAMPLES

• programming “bug”
  • error by programmer leads to failure to write the correct instruction or data
  • this results in a (dormant) fault in code or data
  • upon activation the fault becomes active and produces an error
  • this error may result in failure
EXAMPLES

• Specification related
  • error by a specifier leads to failure to describe a function
  • this results in a fault in a written specification, e.g., incomplete description of a function.
  • this incomplete function may deliver service different from expected service
  • user perceives this as error resulting in failure

EXAMPLES

• Inappropriate human-system interaction
  • inappropriate human-system interaction performed by operator during operation of system
  • results in external fault (from system’s viewpoint)
  • resulting altered processed data is an error...
EXAMPLES

• Reasoning
  • *error* in reasoning leads to a maintenance or operating manual writer’s *failure* to write correct directives
  • results in a *fault* in the manual (faulty directives) that will remain *dormant* as long as the directives are not acted upon...

EXAMPLES

• Combined action of several faults
  • consider trap-door (by-pass access control)
  • this is a development *fault*
  • remains *dormant* until exploited
  • intruder login is deliberate interaction *fault*
  • intruder may create an *error* -> service affected -> *failure*
HARD AND SOFT FAULTS

- Hard (or solid) faults
  - fault activation is reproducible
- Soft (or elusive) faults
  - not systematically reproducible

![Diagram showing hard and soft faults](image)

Fig. 13. Solid versus intermittent faults.

4. DEPENDABILITY AND SECURITY

- From definition point of view

![Diagram showing relationship between dependability and security](image)

Fig. 14. Relationship between dependability and security.
4. DEPENDENCE AND TRUST

• Dependence
  • The dependence of system A on system B represents the extent to which System A’s dependability is (or would be) affected by that of System B.
  • a component a depends upon a component b if the correctness of b’s service delivery is necessary for the correctness of a’s service delivery.

• Trust
  • Trust is accepted dependence.

4. DEPENDENCE AND TRUST

• Levels of dependence
  • from total dependence to complete independence

• Accepted dependence
  • judgement that level of dependence is acceptable
  • judgement possibly explicit, e.g., contract between “parties”
  • judgement may be unwilling, e.g., there is no other option!
  • the extent to which A fails to provide means of tolerating B’s failures is a measure of A’s (perhaps unthinking or unwilling) trust in B.
Availability, integrity, maintainability, reliability, safety, confidentiality...

Don’t think binary, absolute, or deterministic

Do think relative and probabilistic

4.3 ATTRIBUTES OF DEP. & SEC.

- Availability, integrity, maintainability, reliability, safety, confidentiality...
- Don’t think binary, absolute, or deterministic
- Do think relative and probabilistic

Such security policies, relating to a hierarchy of systems—for example, an entire company, its information systems department, and the individuals and computer systems in this department. Separate, albeit related policies, or separate parts of an overall policy document, may be created concerning different security issues, e.g., a policy regarding the controlled public disclosure of company information, one on physical and networked access to the company’s computers. Some computer security policies include constraints on how information may flow within a system as well as constraints on system states.

As with any set of dependability and security specifications, issues of completeness, consistency, and accuracy are of great importance. There has thus been extensive research on methods for formally expressing and analyzing security policies. However, if some system activity is found to be in contravention of a relevant security policy then, as with any system specification, the security failure may either be that of the system, or because the policy does not adequately describe the intended security requirement. A well-known example of an apparently satisfactory security policy that proved to be deficient, by failing to specify some particular behaviour as insecure, is discussed by [44].

Dependability and security classes are generally defined via the analysis of failure frequencies and severities, and of outage durations, for the attributes that are of concern for a given application. This analysis may be conducted directly or indirectly via risk assessment (see, e.g., [25] for availability, [58] for safety, and [32] for security).

The variations in the emphasis placed on the different attributes directly influence the balance of the techniques (fault prevention, tolerance, removal, and forecasting) to be employed in order to make the resulting system dependable and secure. This problem is all the more difficult as some of the attributes are conflicting (e.g., availability and safety, availability and confidentiality), necessitating that trade-offs be made.

4.4 Dependability, High Confidence, Survivability, and Trustworthiness

Other concepts similar to dependability exist, such as high confidence, survivability, and trustworthiness. They are presented and compared to dependability in Fig. 15. A side-by-side comparison leads to the conclusion that all four concepts are essentially equivalent in their goals and address similar threats.

Fig. 15. Dependability, high confidence, survivability, and trustworthiness.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Dependability</th>
<th>High Confidence</th>
<th>Survivability</th>
<th>Trustworthiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>1) ability to deliver service that can justifiably be trusted</td>
<td>consequences of the system behavior are well understood and predictable</td>
<td>capability of a system to fulfill its mission in a timely manner</td>
<td>assurance that a system will perform as expected</td>
</tr>
<tr>
<td>Threats present</td>
<td>1) development faults (e.g., software flaws, hardware errata, malicious logic)</td>
<td>• internal and external threats</td>
<td>1) attacks (e.g., intrusions, probes, denials of service)</td>
<td>1) hostile attacks (from hackers or insiders)</td>
</tr>
<tr>
<td></td>
<td>2) physical faults (e.g., production defects, physical deterioration)</td>
<td>• naturally occurring hazards and malicious attacks from a sophisticated and well-funded adversary</td>
<td>2) failures (internally generated events due to, e.g., software design errors, hardware degradation, human errors, corrupted data)</td>
<td>2) environmental disruptions (accidental disruptions, either man-made or natural)</td>
</tr>
<tr>
<td></td>
<td>3) interaction faults (e.g., physical interference, input mistakes, attacks, including viruses, worms, intrusions)</td>
<td></td>
<td>3) accidents (externally generated events such as natural disasters)</td>
<td>3) human and operator errors (e.g., software flaws, mistakes by human operators)</td>
</tr>
</tbody>
</table>
5.1 FAULT PREVENTION

• General engineering
  • e.g., prevention of development faults
  • development methodologies
  • SW: e.g., information hiding, modularization strongly-typed programming languages
  • HW: e.g., design rules

5.1 FAULT TOLERANCE

• Concepts
  • Diagnosis
  • Rollback recovery
  • Forward recovery
  • Fault masking
  • How are these concepts related?
The measure of effectiveness of any given fault tolerance technique is called its assumption coverage. The measure of the conditional probability that the technique is effective, given that the errors or faults have occurred, or the error and fault handling coverage.

Fault coverage is a critical aspect of dependability. It is evident that not all fault tolerance techniques are equally effective. The measure of effectiveness of any given fault tolerance technique is called its assumption coverage. The conditional probability that the technique is effective, given that the errors or faults have occurred, or the error and fault handling coverage.

Fault tolerance applies to all classes of faults. Protection against intrusions traditionally involves cryptography and through the provision of middleware.

An important issue in coordination of the activities of multiple components is prevention of error propagation. This is known as the fault assumption coverage. Consistent way. This is known as the fault assumption coverage. Consistent way. This is known as the fault assumption coverage. Consistent way. This is known as the fault assumption coverage. Consistent way.

Examples of such protection are voter replication, self-checking checkers, "stable" memory for recovery programs, facilities with respect to the fault assumptions, that can be in turn due to either 1) failed component(s) not behaving as assumed, or 2) the error assumption that differs from the faults really occurred, or the error and fault handling coverage.

Typical examples of such mechanisms are voting replication, self-checking checkers, "stable" memory for recovery programs, dedicated communication links. Typical examples of such protection are voter replication, self-checking checkers, "stable" memory for recovery programs, dedicated communication links.

Byzantine faults will result in a higher failure mode coverage: conservative.

Fault tolerance is often facilitated by the addition of support systems specialized for fault tolerance (e.g., software monitors, service processors, dedicated communication links). Typical examples of such protection are voter replication, self-checking checkers, "stable" memory for recovery programs, dedicated communication links.

Fault tolerance is often facilitated by the addition of support systems specialized for fault tolerance (e.g., software monitors, service processors, dedicated communication links). Typical examples of such protection are voter replication, self-checking checkers, "stable" memory for recovery programs, dedicated communication links.
5.3 FAULT REMOVAL

• During Development
  • Verification
    • the process of checking whether the system adheres to given properties, termed the verification conditions
  • Diagnosis
    • diagnosing the fault(s) that prevented the verification conditions from being fulfilled
  • Correction
    • after correction repeat verification: nonregression verification

• Static Verification
  • Verification without actual execution
  • On System:
    • use static analysis
    • theorem proving
  • On Model of system behavior
    • model checking: state transition model
    • e.g., Petri net, state automata
SIDE NOTE

• What is the relationship between Specification and what has been implemented?

• Discussion on mapping in two directions

VERIFICATION APPROACHES

Fig. 19. Verification approaches.
5.4 FAULT FORECASTING

- Predictive approach

- **qualitative evaluation**, aims to identify, classify, and rank the failure modes, or the event combinations (component failures or environmental conditions) that would lead to system failures;

- **quantitative (or probabilistic) evaluation**, aims to evaluate in terms of probabilities the extent to which some of the attributes are satisfied; those attributes are then viewed as measures.