

---

## PART II

# MODELING THE INTERACTION BETWEEN DEPENDABILITY AND SECURITY

- CHAPTER 6 Taxonomy and Framework for Integrating Dependability and Security  
(Jianku Hu, Peter Bertok, Zahir Tari, RMIT University)
- CHAPTER 7 Stochastic Modeling Techniques for Secure and Survivable Systems  
(Kishor Trivedi, Duke University, and Vaneeta Jindal and S. Dharmaraja,  
Indian Institute of Technology—Delhi)
- CHAPTER 8 Integrated Dependability and Security Evaluation Using Game Theory and  
Markov Models  
(Bjarne E. Helvik, Karin Shallhammar, and Svein J. Knapskog, Norwegian  
University of Science and Technology, Trondheim, Norway)
- CHAPTER 9 Scenario Graphs Applied to Network Security  
(Jeannette M. Wing, Carnegie Mellon University)
- CHAPTER 10 Vulnerability-Centric Alert Correlation  
(Lingyu Wang, Concordia University, and Sushil Jajodia, George Mason University)



---

---

# 6 Taxonomy and Framework for Integrating Dependability and Security<sup>1</sup>

CHAPTER

**Jiankun Hu** RMIT University, Melbourne, Australia  
**Peter Bertok** RMIT University, Melbourne, Australia  
**Zahir Tari** RMIT University, Melbourne, Australia

---

## 6.1 INTRODUCTION

With rapidly developed network technologies and computing technologies, network-centric computing has become the core information platform in our private, social, and professional lives. This information platform is dependent on a computing and network infrastructure, which are increasingly homogeneous and open. The backbone of this infrastructure is the Internet, which is inherently insecure and unreliable. With an ever-accelerating trend of integrating mobile and wireless network infrastructure, things become worse. This is because wireless radio links tend to have much higher bit error rates, and mobility also increases the difficulty of service quality management and security control. The increased complexity of the platform and its easy access has made it more vulnerable to failures and attacks, which in turn has become a major concern for society. Traditionally there are two different communities separately working on the issues of dependability and security. One is the community of dependability that is more concerned with nonmalicious faults [1–4], to name one of just a few.

---

<sup>1</sup> This work is supported by ARC Linkage Projects LP0455324 and LP0455234. For further information, please email the authors at: {jiankun, pbertok, zahirt}@cs.rmit.edu.au.

The other is the security community that is more concerned with malicious attacks or faults [5, 6].

Dependability is first introduced as a general concept covering the attributes of reliability, availability, safety, integrity, maintainability, etc. With ever-increasing malicious catastrophic Internet attacks, Internet providers have realized a need to incorporate security issues. Effort has been made to provide basic concepts and taxonomy to reflect this convergence [7–9]. The original integration effort was to form a joint committee on “Fundamental Concepts and Terminology” by the Technical Committee (TC) on Fault-Tolerant Computing of the IEEE CS, and the IFIP WG 10.4, “Dependable Computing and Fault Tolerance” [7, 10]. Security has been added as an attribute of the class of intentional malicious faults in the taxonomy of faults [10]. Avizienis et al. [7] has provided a very comprehensive set of basic concepts and taxonomy of dependable and secure computing. Jonsson [9] has proposed a system model that views environmental influence as the system input, and system behavior as the system output.

Measures for security and dependability have also been discussed. Based on the work by Avizienis et al. [7], and Jonsson [9], we propose a framework that can generically integrate dependability and security. This chapter does not intend to cover every detail of dependability and security. It places major relevant concepts and attributes in a unified feedback control system framework and illustrates the interaction via well-established control system domain knowledge. Furthermore, the framework has included discussions on lower-level security techniques, such as techniques for confidentiality, authenticity, etc., which have not been addressed in depth in prior work [7, 9]. In this chapter, Section 6.2 provides basic concepts and related work, proposed framework is given in Section 6.3, taxonomy and illustration of the major concepts and attributes under the proposed framework are provided in Section 6.4, and Section 6.5 provides a discussion on the means to attain dependability and security.

## 6.2 BASIC CONCEPTS AND RELATED WORK

In this section, we present basic concepts relevant to the discussion of dependable and secure computing. We also present relevant work in this field.

### 6.2.1 Dependability

Traditionally, dependability is defined as the users’ justifiably trustworthiness on the ability of a system delivering the service to the users [7, 11, 12]. An alternate

definition of the dependability of a system is a system’s ability to avoid service failures that are more frequent and more severe than is acceptable [7]. Although there exist many different ways describing dependability, a consensus view is to describe dependability via threats, attributes, and means. A typical top-level ontology of the dependability is shown in Figure 6.1 [12].

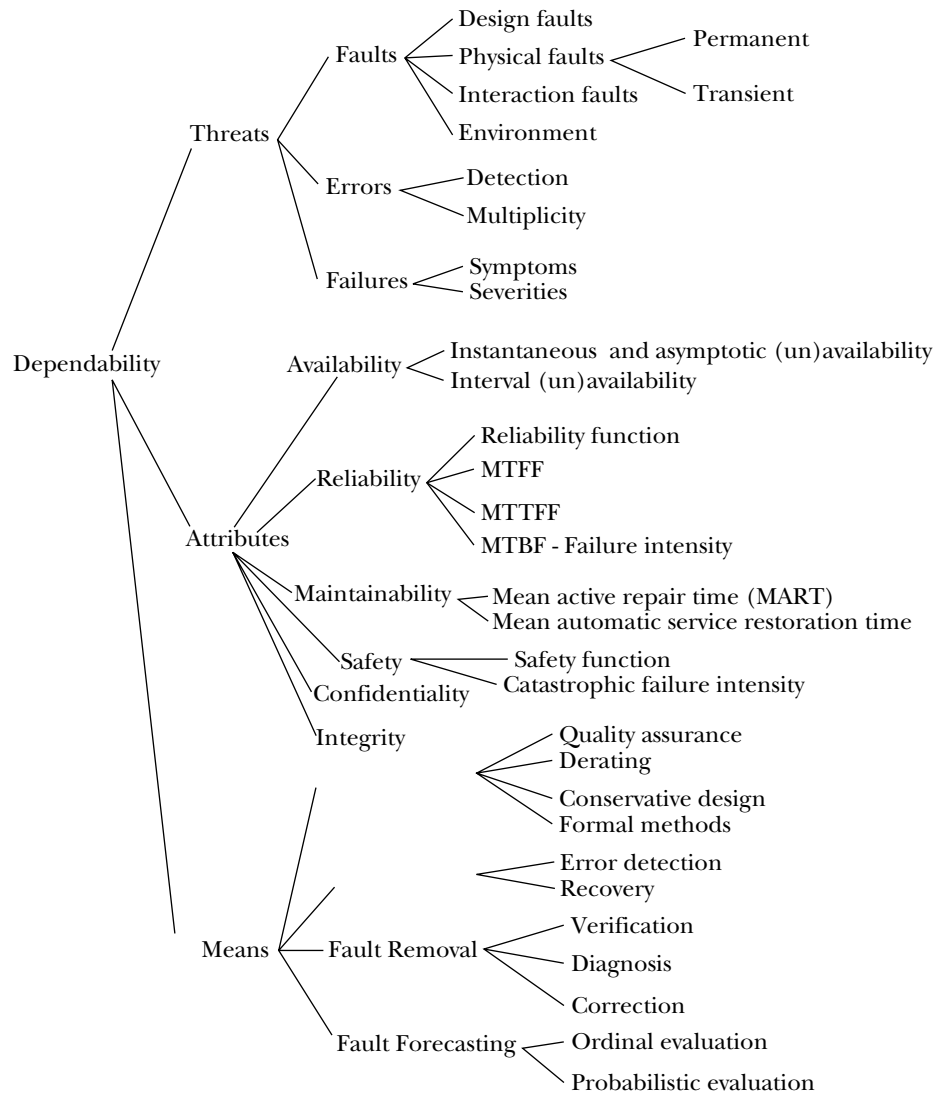


FIGURE 6.1 Dependability ontology based on concepts and terms from IFIP WG 10.4. From Helvik [12].

Under this ontology, faults, errors, and failures are causes of producing threats to dependability that may arise from physical imperfections in a system, external disturbance, mistakes made during specifications, designs, development, operational activities, etc. Threats can also be intentional and hostile. Means are mechanisms to avoid and reduce threats plus recovery from the consequence of threats. Attributes refer to characteristics with respect to dependability, which include [12]:

1. *Availability*: Readiness to provide a set of services at a given time.
2. *Reliability*: Ability to provide uninterrupted service.
3. *Safety*: Ability to provide services without the occurrence of catastrophic failures.
4. *Maintainability*: Ability of a system to support fault removal and service restoration and undergo change.
5. *Confidentiality*: Ability to prevent unauthorized access to and/or handling of information.
6. *Integrity*: The absence of improper alterations of information.

This framework has made a major step in classifying dependability issues and also made an attempt of integration of security and dependability. However, security issues are still treated as a rather standalone component and have not been adequately addressed.

## 6.2.2 Integration of Dependability and Security

In order to integrate dependability and security, one needs to understand essential security issues. Security is a very broad issue even under the context of a networked environment, which is the focus in this chapter. Normally, security has been described via attributes of confidentiality, integrity, and availability [7]. Confidentiality refers to the absence of unauthorized disclosure of information. Integrity refers to the absence of unauthorized alteration of systems or information. Availability refers to readiness for service. Security issues involve many concepts and technologies including cryptography, networking, and intrusion detection. For more background on security techniques, interested readers are referred to Chapter 2 of this book and Stallings [13]. This chapter focuses more on the framework that integrates dependability and security.

Due to the enormous complexity and vast broad areas of dependability and security as well as the rapidly evolving technologies, integrating dependability

and security is a challenging and ongoing effort. Avizienis et al. [7] and Jonsson [9] have proposed a system view to integrate dependability and security that uses system function and behavior to form a framework. A schema of the taxonomy of dependable and secure computing is proposed, as shown in Figure 6.2 [7].

Although this taxonomy and framework are discussed under the context of a system interacting with its environment, there seems to be a lack of a cohesive and generic integration. Jonsson [9] has made an attempt to provide a more generic integration framework by using an input-output system model. In this scheme, faults are introduced as inputs to a system and delivery of service and denial of service are considered as system outputs. However, it is still difficult to illustrate the interactions among many other components. Overall, both system models proposed by Avizienis et al. [7] and Jonsson [9] are open-loop systems that are unable to provide a comprehensive description of the interaction relationship. For instance, there is no mechanism showing the relationship between fault detection and fault elimination in a system. In Section 6.3, we propose a new framework to address these issues.

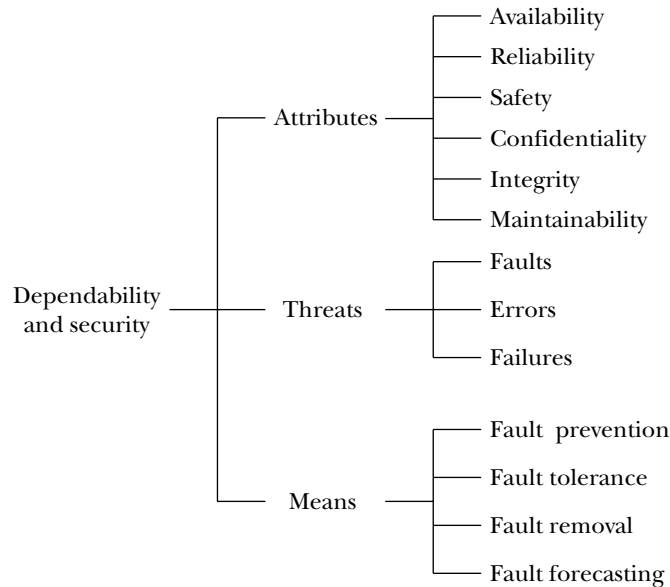


FIGURE 6.2 The dependability and security tree. From Avizienis et al. [7].

## 6.3 PROPOSED TAXONOMY AND FRAMEWORK

In this section, we propose a feedback control system (Figure 6.3) as a framework that can generically integrate dependability and security. Key notations and concepts for the illustration of this framework are also provided.

### 6.3.1 Key Notations of the Feedback Control System Model

The following are conventional notations of feedback control systems. They are tailored whenever needed for our framework.

**Control system:** A system that is under control, normally under regulators’ control, to achieve the desired objectives.

**Desired trajectory:** Desired objectives normally specified by the user.

**Disturbance:** Anything that tends to push system behavior off the track is considered a disturbance. A disturbance can occur within a system or from the external environment.

**Feedback:** Use of the information observed from a system’s behavior to readjust/regulate the corrective action/control so that the system can achieve the desired objectives.

**Feedback control system:** A control system that deploys a feedback mechanism. This is also called a closed-loop control system.

**Filter:** A mechanism retrieving a system’s state to deliver output perceived by the user.

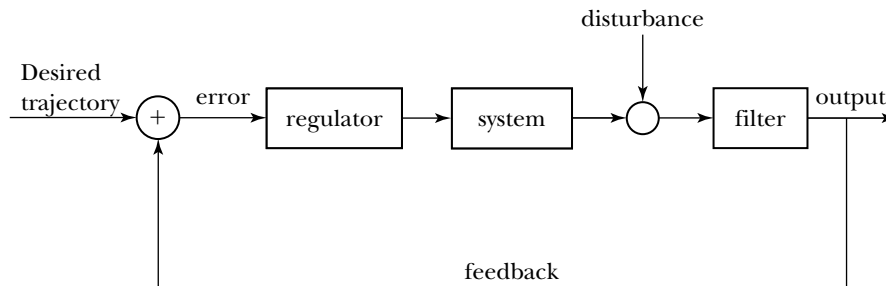


FIGURE 6.3 Feedback control system.

**Open-loop control system:** A control system without a feedback mechanism.

**Regulator:** A mechanism that can combine the input/users’ instructions and feedback information to take corrective control actions to make a system’s behavior achieve its desired objectives.

**System:** A composite constructed from functional components. The interaction of these components may exhibit new features/functions that none of the composite components possess individually.

**System output:** System behavior perceived by the user.

### 6.3.2 Definitions of Basic Concepts of Dependability and Security within the Proposed Framework

**Correct service:** Delivered system behavior is within the error tolerance boundary.

**Desired service:** Delivered system behavior is at or close to the desired trajectory.

**Error:** Deviation of system behavior/output from the desired trajectory.

**Error tolerance boundary:** A range within which the error is considered acceptable by a system or user. This boundary is normally specified by the user.

**Fault:** Normally the hypothesized cause of an error is called fault [7]. It can be internal or external to a system. An error is defined as the part of the total state of a system that may lead to subsequent service failure. Observing that many errors do not reach a system’s external state and cause a failure, Avizienis et al. [7] have defined active faults that lead to error and dormant faults that are not manifested externally.

**Service failure or failure:** An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.

## 6.4 DEPENDABILITY, SECURITY, AND THEIR ATTRIBUTES

The original definition of dependability refers to the ability to deliver a service that can be justifiably trusted. The alternative definition is the ability to avoid service failures that are more frequent and severe than is acceptable. The concept

of trust can be defined as accepted dependence, and dependability encompasses the following attributes [7]:

- ◆ *Availability*: Readiness for correct service. The correct service is defined as what is delivered when the service implements a system function.
- ◆ *Reliability*: Continuity of correct service.
- ◆ *Safety*: Absence of catastrophic consequences on the users and environment.
- ◆ *Integrity*: Absence of improper system alterations.
- ◆ *Maintainability*: Ability to undergo modifications and repairs.

Security has attributes of confidentiality, integrity, and availability. In this chapter, it is assumed that a system does have concern about the security and has reasonable security mechanisms in place. Confidentiality, however, is absent from the above interpretation of dependability. Interestingly, other attributes, such as authenticity and nonrepudiation, are not considered in the previous work. Avizienis et al. [7] merged the attributes of dependability and security together, as shown in Figure 6.2. Similarly, the above attributes can be reframed as follows under the proposed framework that is shown in Figure 6.3:

- ◆ *Availability*: Readiness for correct service. The correct service is defined as delivered system behavior that is within the error tolerance boundary.
- ◆ *Reliability*: Continuity of correct service. This is the same as the conventional definition.
- ◆ *Safety*: Absence of catastrophic consequences on the users and the environment. This is the same as the conventional definition.
- ◆ *Integrity*: Absence of malicious external disturbance that makes a system output off its desired service.
- ◆ *Maintainability*: Ability to undergo modifications and repairs. This is the same as the conventional definition.
- ◆ *Confidentiality*: Property that data or information are not made available to unauthorized persons or processes. In the proposed framework, it refers to the property that unauthorized persons or processes will not get system output or be blocked by the filter.
- ◆ *Authenticity*: Ability to provide services with provable origin. In other words, the output can be verifiably linked to a system.
- ◆ *Nonrepudiation*: Services provided cannot be disclaimed later. In our model, once the system provided an output, there is no way to deny it.

Note that authenticity and nonrepudiation have not been addressed before in the security and dependability framework [7, 11, 12]. These two attributes do not fit into conventional attributes of availability, confidentiality, and integrity (ACI) of security. It seems difficult to include authenticity as a part of ACI. However, we observe that the authenticity issue connects to any of the availability, confidentiality, and integrity problems. Hence, it is more appropriate to express authenticity as an intermediate event toward security faults and also a means to achieving security and dependability. Similarly, it is difficult to include the nonrepudiation as part of ACI. However, unlike the authenticity issue, nonrepudiation problems do not necessarily lead to any of problems of availability, confidentiality, and integrity. Therefore, it seems appropriate to classify nonrepudiation as an independent attribute. An expanded security and dependability tree is given in Figure 6.4.

### 6.4.1 Taxonomy of Faults

In the conventional framework, a fault is defined as a cause of an error. Under the proposed approach, a failure is linked to the error that is outside of the

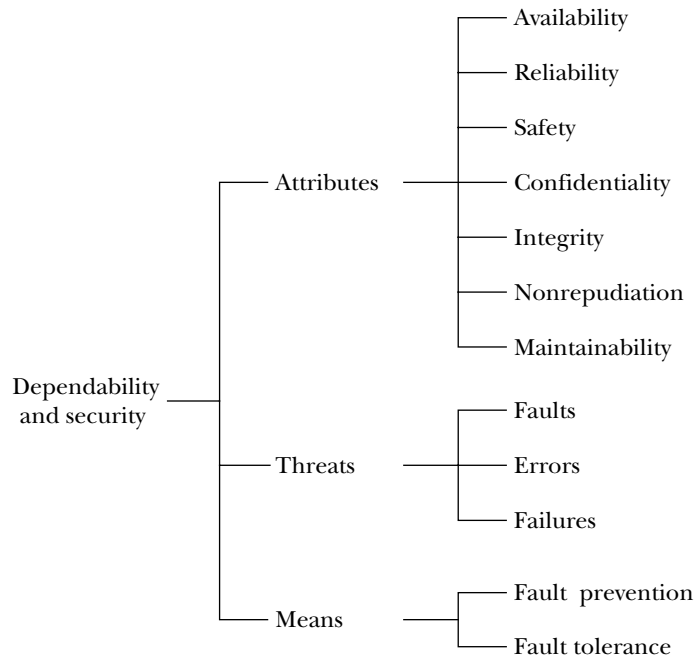


FIGURE 6.4 Expanded dependability and security tree.

error tolerance boundary and is caused by a fault. As for the classification of faults, the most popular method is to categorize them as either malicious or nonmalicious [7].

According to the conventional definition, malicious faults have the objective of altering the functioning of a system during use [7, 9]. Hence “exploit” is classified as operational, external, human-made, software, and malicious interaction fault. Intrusion attempts are also considered as faults. This approach has several flaws. For instance, people often exploit their own system security vulnerability in order to identify security loopholes that do not represent a “malicious objective.” Exploit events are not always faults. Some harmless intrusions that are just designed for fun do not damage a system and do not have malicious objective to interfere with the normal operation of the system. Even if we consider such a fun exercise as malicious, it does not affect the correct service or cause a service error. A fault claim does not fit the definition of faults.

Avizienis et al. [7] has also proposed eight elementary fault classes. However, the combination of these elementary fault classes can generate nonexisting faults. To address this problem, three major partially overlapping groupings, namely, development faults, physical faults, and interaction faults, are introduced in Avizienis et al. [7]. The framework suffers from the problem of classifying nonmalicious activities or error-free activities as malicious faults. We attempt to provide a set of elementary fault classes with minimum overlapping. An intuitive choice is to start with classes that have minimum overlap. We start with two classes, namely, human-made faults (HMF) and nonhuman-made faults (NHMF).

### *HMF*

Human-made faults result from human actions. They include absence of actions when actions should be performed (i.e., omission faults). Performing wrong actions leads to commission faults. Avizienis et al. [7] have categorized human-made faults into two basic classes: malicious faults and nonmalicious faults. They are distinguished by the objective of a developer or of the humans interacting with a system during its use. An exploit activity is classified as malicious fault. As mentioned above, this classification originated from the fault-analysing community and does not integrate well with security. We propose the following new definitions and classifications. HMFs are categorized into two basic classes: faults with unauthorized access (FUA), and other faults (NFUA).

**Faults with unauthorized access (FUA).** This class attempts to cover traditional security issues. We investigate FUA from the perspective of availability, integrity, and confidentiality. Nonrepudiation events normally have the authorized access and hence do not fit in the FUA category.

*FUA and confidentiality.* Confidentiality refers to the property that information or data are not available to unauthorized persons or processes, or that unauthorized access to a system's output will be blocked by the system's filter. Apparently, confidentiality faults fit FUA nicely and can be regarded as a subclass of FUA. Confidentiality faults are mainly caused by access control problems originating in cryptographic faults, security policy faults, hardware faults, and software faults. Cryptographic faults can originate from encryption algorithm faults, decryption algorithm faults, and key distribution methods. Security policy faults are normally management problems and can appear in different forms (e.g., as contradicting security policy statements).

*FUA, integrity, and authenticity.* Integrity is referred to as the absence of malicious external disturbance that causes a system to produce incorrect output. This deviated output can be the result of component failure, but can also be linked to unauthorized access. An integrity problem can arise if, for instance, internal data are tampered with, and the produced output relies on the correctness of the data. Integrity problems are related to but different from authenticity problems, as in the latter case where output produced somewhere else is attributed to the system regardless of correctness. As an example, a person-in-the-middle attack can produce integrity and authenticity faults by altering a message or by producing a totally new one. A confidentiality fault can also occur, if the person-in-the-middle attack gains access to confidential information. This example illustrates that one incident can result in different types of faults.

*FUA and availability.* Availability refers to a system's readiness to provide correct service. Availability faults can be human-made or nonhuman-made. A typical cause of such faults is some sort of denial of service (DoS) attack that can, for example, use some type of flooding (SYN, ICMP, UDP, etc.) to prevent a system from producing correct output. The perpetrator in this case has gained access to a system, albeit a very limited one, and this access is sufficient to introduce a fault. Most viruses and worms also interfere with availability when executing. Some malware that is activated remotely might turn a system into a zombie or sleeping agent. System availability is reduced, sometimes to zero, when these zombies are activated by a perpetrator, when at other times the system is normally available. While availability is affected only temporarily, the fault (i.e., the malware), is continuously present in the system.

Many FUA faults aim at making system output deviate from its desired trajectory and beyond tolerance. At other times, the fault is unintentional (e.g., the result of an operator error). To make a clear distinction between these two cases, we introduce a new concept not discussed elsewhere: malicious attempt fault.

Malicious attempt fault has the objective of damaging a system. A fault is produced when this attempt is combined with other system faults. From the perspective of elementary security attributes—availability, confidentiality, integrity, and nonrepudiation—we classify malicious attempt faults according to their aims as:

1. Intention to disrupt service (e.g., DoS attack).
2. Attempt to access confidential information.
3. Intention to improperly modify a system.
4. Having gained services.

Note that a malicious attempt fault is not a real fault unless it is combined with other faults.

**NFUA.** There are human-made faults that do not belong to FUA. Most of such faults are introduced by error, such as configuration problems, incompetence issues, accidents, etc. Fault detection activity, including penetration testing, is not considered to be a fault itself, as it does not cause system output to deviate from its desired trajectory. Nonrepudiation fault also belongs to the NFUA category, as it normally has an authorized access.

*Nonhuman-made faults (NHMF).* NHMF refers to faults caused by natural phenomena without human participation. These are physical faults caused by a system’s internal natural processes (e.g., physical deterioration of cables or circuitry), or by external natural processes. The latter ones originate outside a system but cross system boundaries and affect the hardware either directly, such as radiation, or via user interfaces, such as input noise [7]. Communication faults are an important part of the picture. They can also be caused by natural phenomena. For example, in communication systems, a radio transmission message can be destroyed by an outer space radiation burst, which results in system faults, but has nothing to do with system hardware or software faults. Such faults have not been discussed before in the existing literature.

From above discussions, we propose the following elementary fault classes, as shown in Figure 6.5. From these elementary fault classes, we can construct a tree representation of various faults, as shown in Figure 6.6.

Figure 6.7 shows different types of availability faults. The Boolean operation block performs either “Or” or “And” operations or both on the inputs. We provide several examples to explain the above structure. We consider the case when the Boolean operation box is performing “Or” operations. F1.1 (a malicious attempt fault with intent to availability damage) combined with

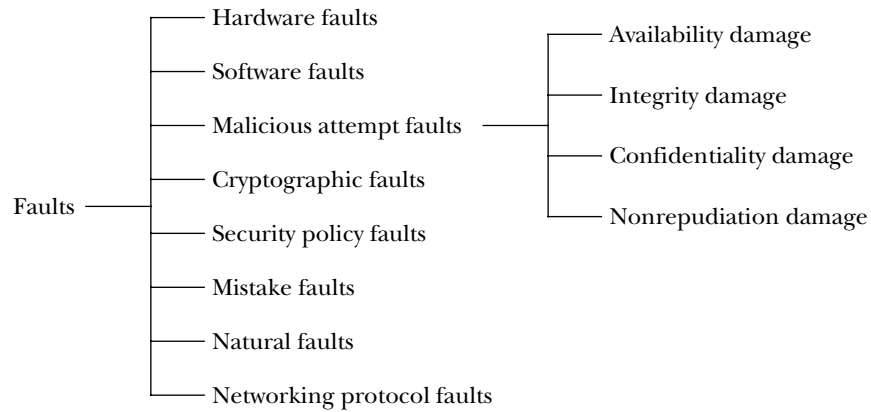


FIGURE 6.5 Elementary fault classes.

6.5

software faults will cause an availability fault. A typical example is the Zotob virus that can lead to shutting down the Windows operation system. It gains access to the system via a software fault (buffer overflow) in Microsoft’s plug-and-play software, and attempts to establish permanent access to the system (back door). F1.1 in combination with hardware faults can also cause an availability fault. F7 (natural faults) can cause an availability fault. F1.1 and F8 (networking protocol) can cause a denial of service fault. Figure 6.8 shows the types of integrity faults.

The interpretation of S2 is similar to that of S1. The combination of F1.2 and F2 can alter the function of the software and generate an integrity fault. Combining F1.2 and F4 can generate a person-in-the-middle attack and so on. Figure 6.9 shows types of confidentiality faults.

The interpretation of S3 is very similar to those of S1 and S2. Combination of F1.3 and F2 can generate a spying type of virus that steals users’ logins and passwords. It is easy to deduce other combinations.

Now let us look at the complex case of a Trojan horse. The Trojan horse may remain quiet for a long time or even forever, and so it will not cause service failure during the quiet period. This is hard to model by conventional frameworks. Within our framework, we need to observe two factors first for the classification. The first factor is the consequence of introducing the Trojan horse, that is, whether it causes a fault or combination of faults, such as availability, integrity, and confidentiality faults. If there is no consequence (i.e., no service deviation error) after introducing it then it is not considered as a fault. This conforms to the basic definition of faults. The second factor is whether the intrusion belongs to

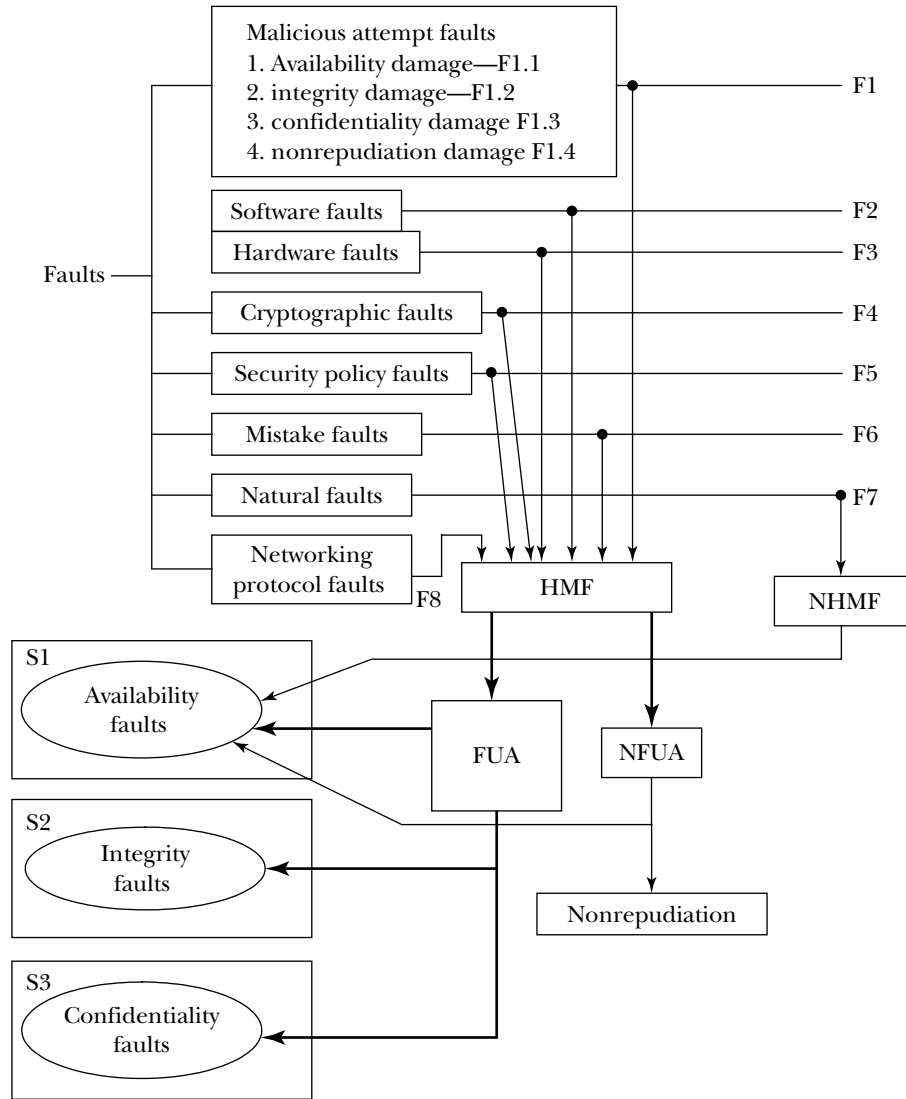


FIGURE 6.6 Tree representation of faults.

6.6

a malicious attempt. Apparently, a network scan by the system administrator is not considered as a fault. When the objective of a Trojan horse is not malicious and it never affects system service, it is not considered as a fault in our framework. Such scenarios have not been addressed properly in many other frameworks where

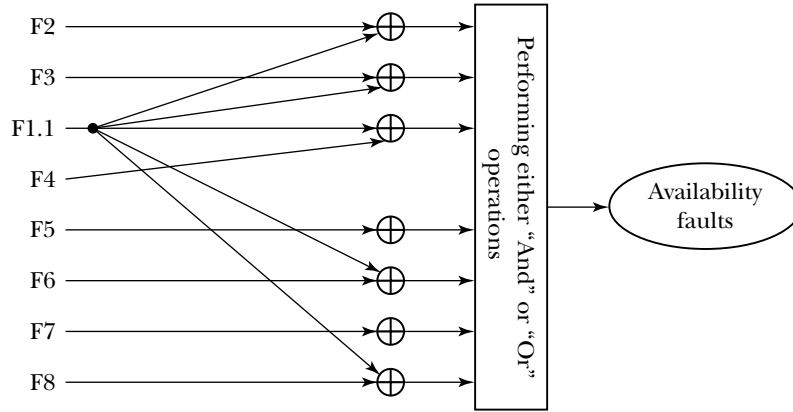


FIGURE 6.7 Detailed structure of S1.

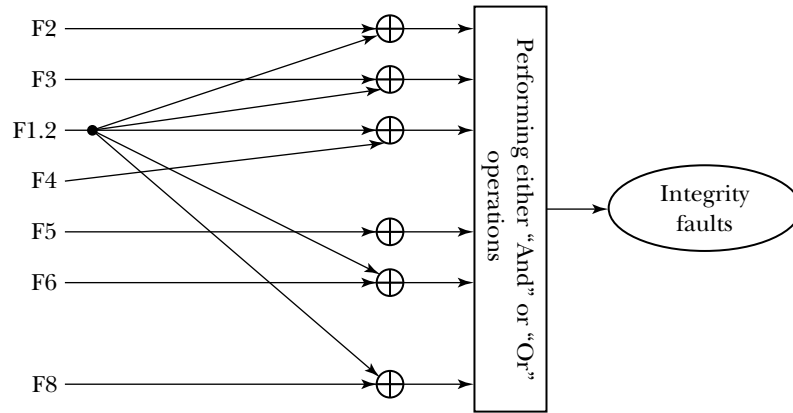


FIGURE 6.8 Detailed structure of S2.

exploit-type activities are characterized as faults even though they may never cause service deviation. If, however, a Trojan horse has a malicious attempt fault and does cause service deviation, then it is considered as a fault classified by S1, S2, and S3 components.

Because a service failure is mainly due to faults, we concentrate our discussion on faults and means to attain fault prevention, fault tolerance, fault detection, and fault removal in this chapter.

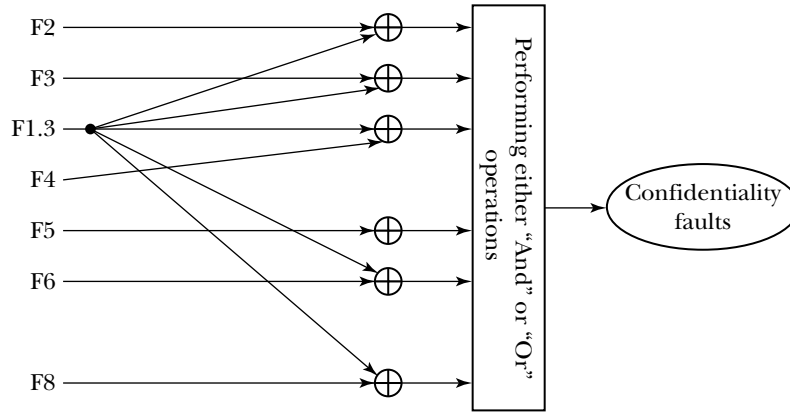


FIGURE 6.9 Detailed structure of S3.

## 6.5 THE MEANS TO ATTAIN DEPENDABILITY AND SECURITY

### 6.5.1 Fault Prevention

Fault prevention is a very general engineering practice and encompasses very broad areas including software, hardware, etc. A comprehensive discussion of this issue is beyond the scope of this chapter. Prevention is better than “curing,” which is also true here. In security-related faults, prevention is paramount. In reducing confidentiality faults and integrity faults, fault prevention is a must and perhaps the only effective mechanism as it faces an unknown proactive effort to create confidentiality and integrity faults. This effort may be made by the most knowledgeable people with enormous resources (e.g., the National Security Agency). The most effective fault prevention mechanism to reduce confidentiality and integrity faults is the deployment of powerful cryptographic schemes. This refers to minimizing cryptographic faults F4. It is also important to manage cryptographic schemes (F5) to further reduce the faults. Cryptographic schemes can provide encryption, that is, a mathematical transformation of the information into an unintelligible form. Only legitimate users with a decryption key can decrypt the encrypted information, which provides confidentiality. As shown in Figure 6.3, it is equivalent to blocking the system output by using the filter. Only users with authorized access to the filter can get access to the information needed.

Although an encryption/decryption mechanism is most powerful and also a required component in providing confidentiality, encryption alone is not enough

to provide integrity faults prevention. This is because in a network environment, people can still cause integrity faults even though they do not have the cryptographic key. A typical case is a replay attack, in which previous legitimate messages are replayed to cause integrity faults even though the attackers do not have the cryptographic key. Authentication and related digital signatures are very effective schemes addressing these issues.

Authentication is a technique used to verify the communicating party for who it claims to be. Digital signature is based on authentication techniques and mainly used for nonrepudiation purposes. Similar to handwritten signatures, a digitally signed message is expected to meet the following requirements [13]:

1. The recipient can verify the claimed identity of the sender.
2. The sender cannot later repudiate the contents of the message.
3. The recipient cannot possibly have concocted the message him- or herself.

For more details of cryptography, network security, access control, etc., and their latest development, see Chapters 2 and 3.

In general, cryptographic-based authentication techniques have a built-in weakness. They are all based on the possession of a certain cryptographic key. This will lead to several security issues. First, the protocol can only verify that the communicating party is the one who has possessed the right key. However, it cannot verify that the communicating party is the genuine user. Secondly, this key is hard to maintain without any risk of leakage and also can be lost. When human beings are involved, biometric authentication seems to be an excellent solution. Biometrics refers to the automatic identification of a living person based on physiological (fingerprint, face, hand geometry, iris) and/or behavioral (voice, signature, keystroke dynamics) characteristics. Biometric identification is preferred over traditional methods involving passwords and PINs (personal identification numbers) for various reasons, mainly that biometric information cannot be used by anyone other than the individual, it cannot be lost or forgotten, it can be used by an illiterate person, and, in conjunction with smart cards, biometrics can provide strong security.

Various types of biometric systems are being used for real-time identification [14–18]. Of all the biometric techniques, fingerprint-based authentication is one of the most mature and proven [14]. Most of biometric authentication protocols are virtually matching issues in the sense that they care about whether the input biometrics match the template biometrics, which does not address other problems such as replay attacks, etc. Han et al. [17] have proposed a hybrid crypto-biometric protocol that can be used in the network environment. This scheme still needs to transfer the key for the recovery of the

fingerprint image, which can be a weak point. The ideal case is to use biometric features directly as cryptographic keys. However, this is still an unresolved challenging issue as biometric features tend to not be precise while conventional cryptography requires accuracy on the cryptographic keys [18]. New security protocols are continuously being developed, and new methods to ensure confidentiality and authenticity under different circumstances are published regularly.

### 6.5.2 Fault Tolerance

A fault will generate a service error. Hence, fault tolerance aims to maintain error within the error tolerance boundary when faults occur, which is also called failure avoidance. Referring to our proposed framework shown in Figure 6.3, fault tolerance means that a system tries to regulate system output so that it keeps track of the desired trajectory even when internal and/or external disturbances are present. It is observed that the fault tolerance in our framework needs a regulator to control system output. The operation of a regulator relies on the error information generated from the feedback. This clearly indicates that fault tolerance needs both error detection and regulation. Although it is not new to consider fault tolerance being composed by both error detection and recovery [7], a generic integration of them as illustrated in this chapter is new. In our framework, error detection and recovery (regulation) are no longer two independent components as proposed in the existing literature [7]. Instead, a regulator needs the error as its input. Our proposed framework provides a better integration of these notions in a generic and seamless way. One common fault tolerance example is router malfunction. When a router error is detected, the system will use the backup router to regulate/recover the system back to the normal operation.

### 6.5.3 Fault Removal

Fault removal and fault forecasting are normally considered separately [7]. Fault removal is considered during system development and system use. During the development phase, fault removal consists of three steps: verification, diagnosis, and correction. During the system use phase, fault removal is considered as corrective or preventive maintenance. Corrective maintenance aims to remove faults that have produced one or more errors that have been reported. Preventive maintenance aims to uncover and remove faults before they might cause errors during normal operation. Fault forecasting is done by evaluating the system behavior with respect to fault occurrence or activation. In this chapter, we do not treat them separately as such. We attempt to integrate them under the proposed unified framework.

We consider that verification, diagnosis, and correction are ongoing and can happen during normal system use. In real-life applications, it is always a good practice to periodically perform verification, diagnosis, and correction. A typical example is the frequent use of “ping” to diagnose network faults during system use. In the proposed framework, sending a “ping” can be interpreted as the input, and the error observed is used to identify faults. For instance, a “not alive” ping reply from a certain, supposedly active server may indicate faults with the server.

System behavior evaluation is also an ongoing process. The evaluation is performed either in passive ways or in proactive ways. In passive ways, system behavior may be evaluated unconsciously by what users perceive. Testing is a typical proactive way of system behavior evaluation. Intrusion (including virus, worms, etc.) detection systems (IDS) always evaluate system behavior to identify intrusions, and may directly block and eliminate the intrusions, or alert the system administrator to do so. Generally, intrusion detection techniques can be classified into two categories: misuse-based intrusion detection and anomaly-based intrusion detection.

#### *Misuse-Based Intrusion Detection*

This technology retrieves intrusions’ features/signatures and establishes a library for the collection of such intrusions. When a particular signature is detected, it is interpreted as an intrusion and is removed from the system. We consider this as evaluating system behavior against a list of predefined system behavior. This can also be called deterministic evaluation. This technique is very effective in detecting known intrusions, but poor in detecting unknown intrusions.

#### *Anomaly-Based Intrusion Detection*

This technique first builds a profile for a system’s normal behavior and then compares operational system behavior with this nominal profile. If a significant deviation is found, an intrusion is announced. This technique is ideal for detecting unknown intrusions but current methods still have a high false rate. The fundamental principle behind such techniques is probabilistic evaluation. Much research effort has been concentrated on the hard-to-detect and malicious intrusions that modify an existing program [19–23]. It involves many PC viruses such as zotob and zombies.

It is interesting to observe that anomaly-based intrusion fault removal is quite similar to fault tolerance, as anomaly-based intrusion detection also needs to detect an error and then use this error to regulate the system output. However, misuse-based intrusion detection is different. It is not confined to detecting an error that has already happened; it can remove faults that have not caused errors yet.

## 6.6 CONCLUSION

In this chapter, a framework has been proposed for the integration of dependability and security. The major contribution of this chapter is to introduce a feedback system to link various concepts and attributes of dependability and security. The framework can help generate relevant taxonomies in a generic and seamless way. Many important concepts and attributes of dependability and security have been illustrated via this framework. Unlike conventional malicious fault classification, we proposed a new concept of malicious attempt fault. This malicious fault is not considered as a real fault unless it combines with other faults that lead to downgrading a system's performance. Such classification provides a more accurate description of dependability and security. Authenticity and nonrepudiation have also been included into the framework. Extensive coverage of low-level techniques has been given under the context of means of achieving fault prevention. Biometric authentication techniques have also been introduced as the latest development in this field. Various intrusion detection techniques have also been addressed as a means of achieving fault removal.

## References

- [1] A. Birolini, *Reliability Engineering, Theory and Practice* (New York: Springer Verlag, 2003).
- [2] G. Buja and R. Menis, “Conceptual Frameworks for Dependability and Safety of a System,” *International Symposium on Power, Electronics, Electrical Drives, Automation and Motion*, Taormina (Sicily)-ITALY May 23–26, 2006, pp. 44–49.
- [3] J. C. Laprie, “Dependable Computing and Fault Tolerance: Concepts and Terminology,” *Proceedings of the 15th IEEE International Symposium on Fault-Tolerant Computing (FTCS-15)*, Ann Arbor, MI June 1985, pp. 2–11.
- [4] J. C. Laprie, “Dependability—Its Attributes, Impairments and Means,” in *Predicting Dependable Computing Systems*, edited by B. Randell, (Berlin; New York: Springer: 1995), pp. 3–24.
- [5] W. Molisz, “Survivability Function—A Measure of Disaster-Based Routing Performance,” *IEEE Journal on Selected Areas in Communications* 22, No. 9 (Nov. 2004): 1876–1883.
- [6] D. Medhi and D. Tipper, “Multi-Layered Network Survivability—Models, Analysis, Architecture, Framework and Implementation,” *Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX'00*, Hilton Head, SC, Jan. 25–27, 2000, pp. 173–186.

- [7] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwhehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Transactions on Dependable and Secure Computing* 1, No. 1 (Jan.–March 2004): 11–33.
- [8] J. J. Clark and W. M. Fitzgerald, “SecurIST: Coordinating the Development of a Strategic Research Agenda for Security and Dependability R&D,” *39th Annual International Carnahan Conference on Security Technologies, CCST’05*, Las Palmas de Gran Canaria, Spain. Oct. 11–14, 2005, pp. 295–299.
- [9] E. Jonsson, “An Integrated Framework for Security and Dependability,” *Proceedings of the 1998 Workshop on New Security Paradigms*, (Charlottesville, Virginia: ACM Press, 1998), pp. 22–29.
- [10] Special Session, “Fundamental Concepts of Fault Tolerance,” *Proceedings of the 12th IEEE International Symposium on Fault-Tolerance Computing (FTCS-12)*, Santa Monica, USA June 1982, pp. 3–38.
- [11] K. Kyamakya, K. Jobmann, and M. Meincke, “Security and Survivability of Distributed Systems: An Overview,” *Proceedings of IEEE Milcom*, Los Angeles, CA, May 2000, pp. 449–454.
- [12] B. E. Helvik, “Perspectives on the Dependability of Networks and Services,” *Teletronikk* 3 (2004): 27–44.
- [13] W. Stallings, *Cryptography and Network Security: Principles and Practices*, 3rd ed., (Englewood Cliffs, NJ: Prentice Hall, 2003).
- [14] Y. Wang, J. Hu, and D. Philip, “A Fingerprint Orientation Model Based on 2D Fourier Expansion (FOMFE) and Its Application to Singular-Point Detection and Fingerprint Indexing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, No. 4, pp. 573–585 (April 2007).
- [15] Y. Wang, J. Hu, and F. Han, “Enhanced Gradient-Based Algorithm for the Estimation of Fingerprint Orientation Field,” in *Applied Mathematics and Computation*, Vol. 185, No. 2, pp. 823–833, Feb. 2007, (Boston: Elsevier, 2007).
- [16] A. K. Jain, S. Prabhakar, L. Hong, S. and Pankanti, “Filterbank-Based Fingerprint Matching,” *IEEE Transactions on Image Processing* 9, No. 5 (2000): 846–859.
- [17] F. Han, J. Hu, X. Yu, Y. Feng, and J. Zhou, “A Novel Hybrid Crypto-Biometric Authentication Scheme for ATM Based Banking Applications,” *IAPR International Conference on Biometrics (ICB2006)*, Hong Kong, Jan. 5–7, 2006. Published in *Lecture Notes in Computer Science* 3832 (New York: Springer-Verlag, 2005), pp. 675–681.
- [18] F. Han, J. Hu, and X. Yu, “A Biometric Encryption Approach Incorporating Fingerprint Indexing in Key Generation,” *International Conference on Intelligence Computing (ICIC06)*, Kunming, China, 2006. Published in *Computational Intelligence and Bioinformatics, Lecture Notes in Computer Science* 4115 (New York: Springer-Verlag, 2006), pp. 342–351.

- [19] D. Song, M. I. Heywood, and A. N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: An Example from Anomaly Detection," *IEEE Transactions on Evolutionary Computation* 9, No. 3 (June 2005): 225–239.
- [20] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "A Comprehensive Approach to Intrusion Detection Alert Correlation," *IEEE Transactions on Dependable and Secure Computing* 1, No. 3 (July–Sept. 2004): 146–168.
- [21] X. D. Hoang, J. Hu, and P. Bertok, "Intrusion Detection Based on Data Mining," *The 15th International Conference on Enterprise Information Systems*, Vol. 3, Angers, France, 2003, pp. 341–346.
- [22] X. D. Hoang, J. Hu, and P. Bertok, "A Multi-Layer Model for Anomaly Intrusion Detection," *Proceedings of IEEE International Conference on Networks*, vol. 1, Atlanta, Georgia, Sept. 2003, pp. 531–536.
- [23] X. D. Hoang and J. Hu, "An Efficient Hidden Markov Model Training Scheme for Anomaly Intrusion Detection of Sever Applications Based on System Calls," *Proceedings of IEEE International Conference on Networks*, Vol. 1, Singapore, 2004, pp. 470–474.