On the Functional Relation Between Security and Dependability Impairments

Erland Jonsson, Lars Strömberg, and Stefan Lindskog
Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
SWEDEN
email: erland.jonsson/larst/stefanl@ce.chalmers.se

ABSTRACT

Problems related to security and dependability/ reliability are still treated separately in many contexts. It has been shown that there is a considerable conceptual overlap, however, and an integrated framework to the two disciplines has already been suggested. This paper shows that there is also a conceptual overlap of impairments from these areas and suggests an integrated approach that clarifies the functional relation between these, both from dependability and security viewpoints. The overall objective is to arrive at a general and clear-cut framework that would describe how trustable (dependable, secure) a system is, regardless of the reason for its not being totally trustable. For example, it should be possible to treat a system failure caused by an intentional intrusion or a hardware fault using the same methodology. A few examples from real-world situations are given to support the suggested approach.

Keywords: Security, Dependability, Impairment, Threat, Intrusion, Vulnerability, Modelling, Terminology.

1. INTRODUCTION

It has been shown that there is a considerable conceptual overlap between the two disciplines of security and dependability and that it may be fruitful to view them as different aspects of a single "meta-concept", as detailed in [20] and summarized in Section 2. The present paper demonstrates that a similar problem exists at lower hierarchical levels and suggests a way towards a unification of conceptual and terminological discrepancies at these levels.

An illustration of such discrepancies is that, in the dependability discipline, reasons for *failures* are called *faults* and *errors*, whereas security people traditionally talk about *attacks* that cause *breaches* and *vulnerabilities*. The extent to which these terms correspond is not immediately clear, even if there seems to be similarities. Other questions of this type can be posed. What are the relations between e.g. fault, attack, flaw, error, bug, vulnerability, defect and violation? Do some of these terms represent identical concepts? Should we in that case look for a unified terminology, or is it justifiable to maintain separate terminologies for each discipline? These are questions that must be answered as integration work proceeds. While a full answer is not given in this paper, we make a first attempt towards a unified approach that we hope will facilitate further work in this direction.

Finally, it must be stressed that the work presented in this paper is about new concepts and that it also uses old concepts in a new way. In general, new concepts call for the invention of new terms or redefinition of old terms, since it is essential that the concepts can be properly addressed and understood. It would be expected that the person who suggests new concepts would also suggest a corresponding terminology and that he clarifies the relations with the established usage of the terms. Although we have tried to do this, we realize that re-defining words or changing the usage of words is quite a delicate task with little prospect of being successful. Therefore, we do not wish to strongly defend any part of the *terminology* in this paper. The underlying *concepts* have our full support, on the other hand, and we believe that, once these concepts become commonly accepted, the issue of proper terminology will find its solution.

In the following, Section 2 summarizes the integrated conceptual framework for security and dependability. In Section 3, the same exercise is carried out for security and dependability impairments. Some illustrative examples are given in Section 4. Section 5 concludes the paper and discusses possible future directions of work.

2. DEPENDABILITY AND SECURITY

2.1 Traditional dependability and security concepts

Dependability is defined as an "umbrella concept" with four attributes: *reliability, availability, safety* and *security* [22]. The relation between these attributes, in particular the three first ones, and fault-tolerance has been discussed at some length in the literature [2], [4], [10], [14], [15], [16], [24], [25], [26], [27], [28]. There is a high degree of consensus on their meaning. Reliability is a characteristic that reflects the probability that the system will deliver its

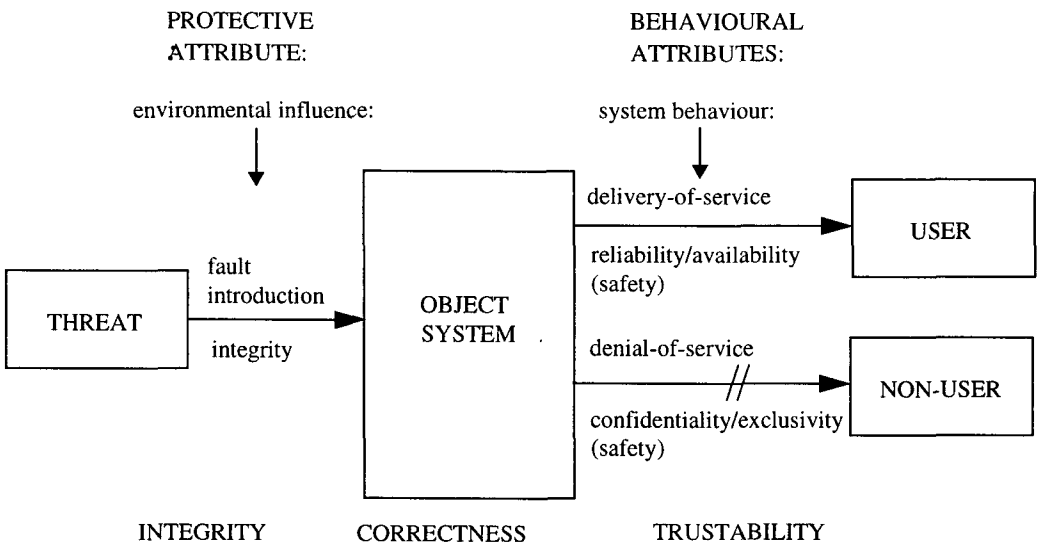

Figure 1. The system model

service under specified conditions for a stated period of time, whereas availability reflects the probability that the system will be available, or ready for use, at a certain moment in time. Safety denotes the system's ability to fail in such a way that catastrophic consequences are avoided. Thus, safety is reliability with respect to catastrophic failures. Security refers to the system's ability to prevent unauthorized access or handling of information and to its ability to withstand illegal interaction or attacks against such system assets as data, hardware or software. The notion of security normally assumes a hostile action by a person, the attacker.

Among security people, security is further decomposed into three different aspects: *confidentiality, integrity* and *availability* [13], [29]. Please note that availability comes in twice, first as an attribute of dependability and then as a sub-attribute of security. However, the definitions agree in these two cases. Confidentiality is the ability of the computing system to prevent disclosure of information to unauthorized parties. Integrity is the ability of the computer system to prevent unauthorized modification or deletion.

2.2 An integrated framework for dependability and security concepts

To clarify the conceptual overlap above, an integrated approach to security and dependability was suggested in [20]. This paper makes an interpretation of security and dependability concepts in such a way that they can be treated simultaneously. According to this interpretation, the object system interacts with the environment in two, basically different ways. The object system either takes an input from the environment or delivers an output or service to the environment, see Figure 1.

The environmental influence may consist of many different types of interactions. The type of interaction we are interested in here is that which involves an introduction of faults into the system. From a security viewpoint, intentional and often malicious faults, i.e. security breaches, are particularly interesting. Since faults are detrimental to the system, we seek to design the system such that the introduction of faults is prevented. We denote this ability ***integrity***. It encompasses the protective[1] aspect of security/dependability. This definition agrees well with a common understanding of the word in the security community, although somewhat generalized. However, there exist several different meanings of the word integrity in various other contexts [29].

The output from the system includes the service delivered by the system to the users. We call this the system behaviour. There are two different types of users: the authorized user, denoted User, and the unauthorized user, denoted Non-user. The desired (and preferably specified) delivery-of-service to the User can be described by the behavioural attributes of reliability and availability. Less often specified, but still desired, is that the system shall have an ability to deny service to the Non-user, denial-of-service. Note that this is a generalization of an existing concept. Normally, and taking the viewpoint of the system owner, the term represents an unwanted behaviour of the system with respect to the User, and thus a violation of the specification. In our model it can also be referred to the Non-user, but in this case the denial-of-service is a specified and desirable behaviour.

Denial-of-service with respect to the Non-user is described by the behavioural attributes of confidentiality (for information) and exclusivity (for use). In both these cases, safety denotes a disruption of the denial-of-service, i.e. a delivery-of-service, that would lead to catastrophic consequences. We suggest the use of the word ***trustability*** for the aggregate of behavioural attributes. A trustable system should be reliable, available, safe etc. Although the word trust-

[1] In the paper referred to, we used the term "preventive", but "protective" seems to be a more adequate word.

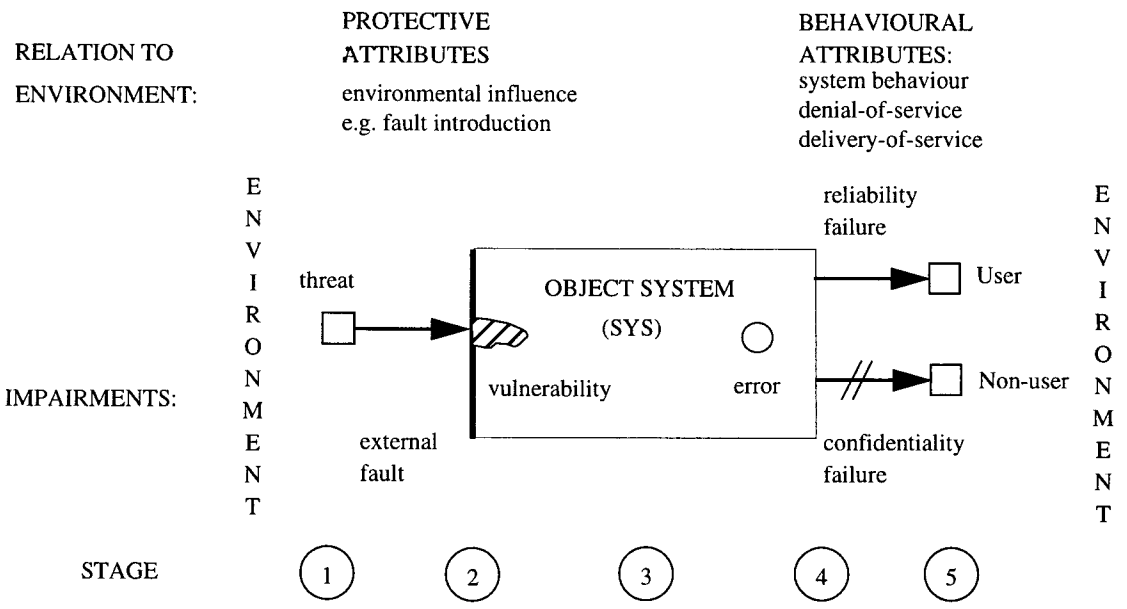

Figure 2. The stages of security/dependability impairments

ability might indicate that the model aims at some kind of subjective belief, this is not our intention. The model is intended to be objective and "technical". As was pointed out in the introduction, terminology is a difficult issue, and the reader is invited to suggest alternate terms.

The third attribute that may apply to a system is ***correctness***, which would denote that the system is free from unwanted internal states, errors, and free from vulnerabilities. Thus, the meta-concept of security/dependability—we refrain from suggesting a name here—can be split up into the attributes of integrity, correctness and trustability, referring to system input, internal state and output, respectively.

3. INTEGRATING SECURITY AND DEPENDABILITY IMPAIRMENTS

3.1 A refined system model

A model of a dependable system contains at least two components, the *object system* (SYS) and the *environment.* Consider a specific system as depicted in the block diagram in Figure 2. Here, a circle denotes a state, an arrow an event and a square a (sub)system. Circled numbers refer to Table 1 in paragraph 3.10.

The block diagram describes the dependability impairments we shall discuss. The environment interacts with the SYS by generating inputs to it and by receiving outputs from it.

The discussion in this section starts from the observation that a failure is normally preceded by a chain of events that leads to that failure. These events and their intermediate effects on the system are called *impairments.* We suggest definitions of impairments that are adapted to both the traditional dependability and traditional security domains.

3.2 Existing definitions of impairments

There exists a number of various usages of the terms for dependability impairments: faults, errors and failures. For example, the fault-tolerance community and the software community have different opinions of causal direction between faults and errors, as pointed out in [12]. Thus, the software community claims a fault is the result of a programmer's error, whereas a fault is the reason for an error in the fault-tolerance community. This is discussed in some detail in [17]. A third alternative for the definition of fault is found in [27] and, in that document, the terms error and failure are not covered at all. Other relevant suggestions and discussions are found in e.g. [4], [5], [6], [7], [9] and [10].

A similar approach to that of the software community is found in a work on classification of software vulnerabilities [21]. Here, an error is defined as a mistake by the programmer and a fault as the resulting incorrect code.

In the security community the term vulnerability is often related to the notion of security policy [21]. A vulnerability is something that makes it possible to break the security policy. A specific system can have different vulnerabilities depending on the security policy context. Thus, Bishop defines a vulnerable state as "an authorized state from which an unauthorized state can be reached using authorized state transitions" [8], and a vulnerability is the direct reason that the system is in a vulnerable state.

3.3 System states

The term *system* is used here in a very general way. A system can be composed of a set of interacting components, each in itself a (sub)system. The aggregate of components with which the system directly interacts is called the environment of the system. A special part of a system is the *system documentation.* A system is created as soon as the first document referring to the system is made. This means that the early phases of system development are also included in the *system life.* Faults can enter the system at any time, causing the system to be in an erroneous state.

We use the term state (e.g. error state) to denote a class of states in which a system can be. Membership in a state class is determined by a function mapping a system state to a boolean value. If the state function is only a function of the internal variables of the system, the state is *internal*; otherwise, i.e. if it is also a function of the environment influence, it is *external*.

An error state is internal. A transition into an error state can be triggered by a cause that is internal or external to the system. If the system allows a transition to an error state by an external cause, there exists a vulnerability in the system, i.e. the system is in a vulnerable state. A vulnerable state is a subset of a correct state, the other alternative being correct and non-vulnerable. An error state can *propagate* into another error state—once or several times. The propagation may (or may not) proceed until it reaches the system boundary, where it manifests as a failure. A failure is a state that is defined with respect to the system environment. The system is in a failed state when it does not behave according to its specification.

3.4 Threat

In theory, all subsystems in the environment may interact with the SYS. This interaction may be intentional in the sense that the subsystem is functionally connected to the SYS. The interaction may also be unintentional, reflecting no functional relationship. The interaction, whether intentional or unintentional, may result in undesired effects to system correctness and/or trustability. Thus, from this viewpoint, the environmental subsystem represents a *threat* to both the dependability and the security of the SYS.

Definition: A **threat** is an environmental subsystem that can possibly introduce a fault in the system.

The notion of threat has normally been linked to intentional faults and the security attribute. The above definition is much broader. Any subsystem in the environment may constitute a threat to the system.

3.5 Vulnerability

The critical points in a system are the places where faults are introduced, which for external faults are at the boundary between the environment and the SYS. The environment contains the threats, whose behaviour represent a risk for fault introduction. This risk can never be completely eliminated, and there will always be a remaining probability for external fault introduction into the system. Thus, it is always worthwhile to improve the system in such a way that it can better withstand the threats. We define the term vulnerability:

Definition: A **vulnerability** is a place where it is possible to introduce a fault.

A vulnerability can be located in e.g. the code, the configuration or the design of the system. The presence of a vulnerability means that the system is in a *vulnerable state*.

In principle, all systems are vulnerable to some extent. Therefore, it should be possible to define the "degree of vulnerability" for a specific system. This can probably be done in terms of probability of exploitation of the breach, under the assumption of a certain environment, or as some kind of "threshold" that must be exceeded in order to successfully attack the system [19] or by some other method.

The vulnerability (deficiency, weakness, flaw) concept is well known from the security domain. A security attack may aim at planting a vulnerability in the system, a vulnerability that can later be exploited by further attacks to cause loss or harm. The term vulnerability is also applicable for non-intentional interaction. For example, a hardware vulnerability can typically be an unshielded cable, which is inclined to pick up external noise.

A significant property of a vulnerability is that it will *not* propagate during normal operating conditions but will function only as a channel for external fault introduction.

3.6 Fault

A fault that can be related to a threat is called an *external fault*, since the source of the fault is found outside the system. *Internal faults* are faults that arise (apparently) spontaneously somewhere in the system, i.e. with no direct relation to a threat. The following definition of fault covers both cases:

Definition: A **fault** is an event leading to an error or a vulnerability in the system.

A fault is an *event* or *system state change* and is regarded as an atomic phenomenon. Thus, a fault is an inherently transient phenomenon and is not permanent. Neither is a fault intermittent. An "intermittent fault" is regarded as a number of repeated transient incidents. The fault is the direct reason for the error occurrence in the system and will lead to an error by definition.

3.7 Attack and breach

An attack is an intentional activity conducted or initiated by a human. If the system is in a vulnerable state an attack may be successful and cause a type of fault called a *breach*. A breach results in an error or a vulnerable state in which the system security policy is violated.

Definition: A **breach** is an external, intentional fault. Thus, a breach causes an error or vulnerability in the system.

Thus, loosely, a breach can be regarded as a "security fault", whereas, in analogy, "normal" faults, that are not the result of some intentional human interaction, could be regarded as "reliability faults". However, we are not convinced that there is a need for such a distinction.

The breach is a result of an attack, leading to the following definition:

Definition: An **attack** is an attempt to cause a breach in the system.

3.8 Error

Definition: An **error** is a system state that may lead to a system failure during normal system operation.

Since this definition is very general and is intended to be applicable to many different types of systems, the word *state* must be understood in a broad sense. Thus we will avoid giving an exact definition or interpretation of the word that would be valid for all cases. Once an error has occurred in the system, the system is *erroneous*. The error may propagate to the system boundary and lead to a failure. An error is the result of either a breach or a reliability fault. We refer to the former as a security error and to the latter as a reliability error.

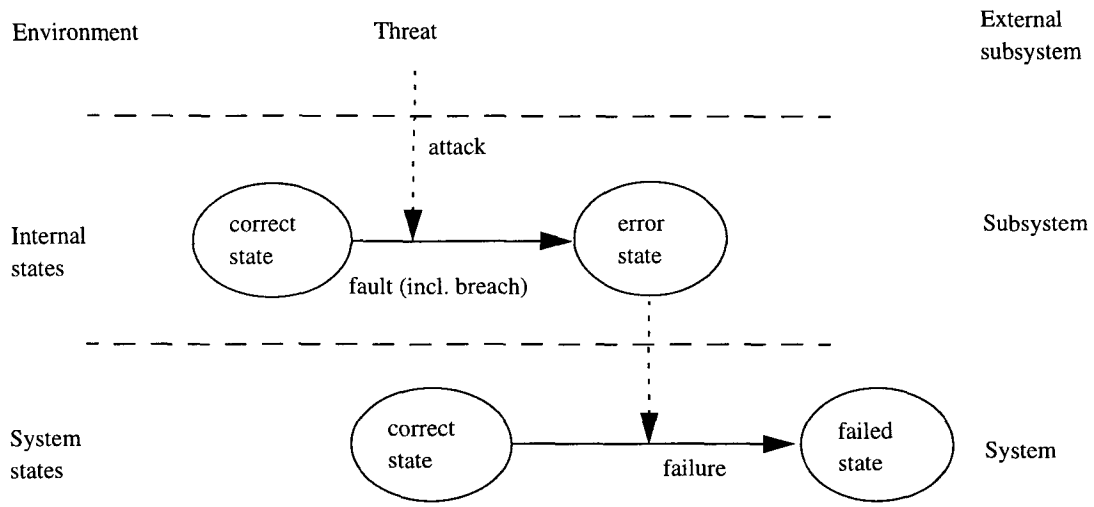

Figure 3. Summary of terminology

It must be noted that the difference between error and vulnerability is sometimes quite subtle, which is illustrated by the examples in Section 4. This is due to the fact that it is not always evident what constitutes "normal behaviour". This, in turn, is dependent on the security policy, which may not be very explicit or perhaps not documented at all. However, the same problem exists for the definition of failure, which is very much dependent on the system specification.

3.9 Failure and failed state

A failure or failure transition is an event that represents a state change of the total system with respect to the environment and the service it delivers. Before the failure occurs, the service delivered by the system is in accordance with the specification. After the failure, the service deviates from what is specified. Thus the following definition applies:

Definition: A **failure** is the event at which a deviation first occurs between the service delivered by the system and the expected service, as defined in the system specification.

A failure is the result of an error—whether a "security error" or a "reliability error"—that has propagated to the system boundary. There are two classes of failures. The first refers to a disruption of the delivery-of-service to the authorized user, informally called *"reliability failure"*, and the second to a disruption of the denial-of-service to unauthorized users, informally called *"confidentiality failure"*. We get the definition:

Definition: A failure that violates the confidentiality property of an object system is a **confidentiality failure**, and a failure that violates the reliability or the availability property of an object system is a **reliability failure**.

Here, a very significant observation is that ***there is no one-to-one correspondence between the error classes and the failure classes***. Thus, a "security error" can lead to a "reliability failure" or a "confidentiality failure". In the same way, a "reliability error" can lead to either type of failure. Section 4 gives a few examples of this. A system that has made a failure transition is in a failed state:

Definition: A system that exhibits a deviation between the delivered service and the specified service is said to be in a **failed state**.

Note that there is a remarkable hierarchical symmetry between the state changes of fault-error and failure-failed state. A fault is an event that transforms the state of a system component (subsystem), i.e. the internal state of the system, from correct to incorrect (erroneous). Similarly, a failure transforms the system state from correct to incorrect (failed). We thus have the same type of state change in both cases but at different hierarchical levels. A natural extension of this would be to apply the same idea further down in the hierarchy, to sub-subsystems, or further up in the hierarchy, if appropriate.

One problem is that the two different failed states are probably quite different. In particular, the notion of "state" in the confidentiality context is a little tricky, and it is not evident how this can be characterized. A confidentiality failure can normally not be "repaired", nor is there an "undo" function, at least not easily. There are also some open issues that refer to the duration of a confidentiality failed state. Once a piece of information has been released inappropriately, the system may work normally and the "hole" may have disappeared. Does this mean that all confidentiality failures are transient? We feel that there is some work to be done here.

Figure 3 summarizes the terminology. If the object system is in a vulnerable state, an attack originating from an external threat can cause a transition to an error state. The error state can in turn cause a transition of the total system from a correct state, i.e. the system delivers service, to a failure state, i.e. the system does not deliver service as specified. It should be noted that the figure is simplified in that it does not show the propagation of error states within. the system

stage	1	2	3	4	5
SYS-ENV interaction	environmental influence	system protection	system operation	system behaviour	impact on environment
impairment propagation	fault generation	fault introduction	error propagation	failure occurrence	failure consequence

Table 1. Types of impairments

3.10 .The relation of impairments to the system and its environment

The development and propagation of impairments and their relation to the system and its environment are summarized in Table 1. The stage numbers refer to Figure 2.

The source of the external fault is the threat in the environment. The threat represents a potential *environmental influence* with respect to the system, performing an act of *fault generation.* The threat may attempt to introduce the external fault into the system, *fault introduction.* The system tries to counter this attempt by means of various methods for *system protection.*

By definition, a fault will lead to an error. The error may or may not start propagating, depending on the operational circumstances, *error propagation.* Therefore, an error may not necessarily lead to a failure and, even if it does, the failure may manifest itself only after a considerable delay. The fundamental observation here is that *it is not until a failure has occurred that any harm is done,* as experienced by the user As a consequence, a fault or an error *will not affect the trustability* of the system *if it never leads to a failure.* Thus a system may be subjected to faults and contain errors and still never exhibit a failure. As a matter of fact, all systems that contain software of any significant size do contain errors. Despite this fact, many of these errors may never propagate to cause a failure. There are investigations that show that many errors will show up as failures only after a delay that is indeed considerable: thousands of years [1].

It should be noted that the error propagation model is not always applicable. This is especially so in some *collapsed* cases, where the failure emerges virtually directly with no significant delay from the fault event. It may even be difficult to define or distinguish the corresponding fault and error(s). Typical examples are failures that are the result of violent action towards hardware, e.g. crashing the screen, but also include many confidentiality failures, such as the overhearing of a message, whether acoustic, visual or electronic.

4. EXAMPLES

This section gives a few examples of functional relations between different impairments, impairment propagation and impact on the environment. We also show how faults can lead to reliability and to confidentiality failures.

4.1 UNIX kbd_mode command

This example shows how a breach ("security fault") or a "reliability fault" could cause a reliability failure."

The *kbd_mode* command is intended to reset the keyboard of a SunOS system to a well-defined state. However, it has turned out that it is possible to execute the command remotely on another machine, in which case the keyboard of that machine becomes locked, i.e. it becomes unavailable to the User.

The fact that the execution of the command leads to something not intended by the designer means there is an error in the software, since it could be expected that executing a command remotely should lead to the same result as executing it locally, or should at least be disregarded by the system. The programmer thus made a fault in the design process, which led to this error. The error is activated by the attacker, who makes it propagate to cause a reliability failure, so that the User can no longer use his machine.

It is also quite clear that a hardware (component) fault may lead to exactly the same result, i.e. a disabling of the keyboard.

4.2 IP stack and buffer overrun problems

This example shows how a "reliability fault" leads to a vulnerability, which is then exploited by an attacker, who performs a breach ("security fault") that leads to a reliability failure.

It has recently become evident that many operating systems, including Windows NT, Windows 95 and many versions of UNIX, do not handle packet header information properly in the IP stack. This has been demonstrated in a number of exploit scripts. *Teardrop, bonk, boink, Land* and *LaTierra* are examples of such scripts. Detailed information on these is given in [11]. When executing one of those, the victim machine will typically crash completely.

In this particular case, the problem is that many operating systems blindly trust the information stored in the IP header of receiving packets. This is the result of a programmer having made a fault that resulted in a vulnerable operating system software. The vulnerability is exploited by the scripts described above, which leads to an error that propagates and causes the machine to crash or degrade. Thus, the failure is a typical reliability failure caused by a security error.

A similar situation exists for buffer overflow attacks. Here, the vulnerability lies in the (code that specifies the) insufficient control of parameters. The *NTCrash* program demonstrates how such a vulnerability can be used for an intentional crash of a Windows NT system [10].

4.3 Hardware faults

This example shows how a "reliability fault" can cause either a "reliability failure" or a "confidentiality failure".

Ionizing radiation is a threat. An ion passing a sensitive depletion layer introduces a fault that is manifested as an inverted bit in a memory, which is the error. In this case the error is "soft" in the sense that recovery can be made using logical means. Thus, if we are lucky, the program will soon clear this bit position and thereby delete the error: the error will not propagate. If we are unlucky, the

bit flip error will propagate and cause a failure. Suppose that the flipped bit was a control signal that enabled encryption of an outgoing message and that the flip made the message go out in clear text (unencrypted). In that case, the failure may result in exposure of secret information to an unauthorized user, i.e. a confidentiality failure.

It is also easy to imagine cases where hardware bit flips would lead to a program crash, i.e. a reliability failure.

4.4 Trojan Horses

This example shows how a breach ("security fault") may cause either a "reliability failure" or a "confidentiality failure".

A User has left his login file world readable and writeable. This vulnerability can be exploited by an attacker to plant a Trojan Horse. This is a breach. The Trojan Horse is activated for a certain set of conditions that may e.g. involve time, certain actions by the User etc. The planted Trojan Horse is an error in the system since it will be activated and perform its task as a result of normal operation, provided certain conditions are fulfilled. This task may be deleting all the User's files on the hard disk, which is a reliability failure.

Another example would be if the Trojan Horse were activated when the User sent email to a certain receiver. The action in this case could be copying the email to the Non-user, which is a confidentiality failure.

4.5 Back door

This example shows how a breach ("security fault") can cause either a "reliability failure" or an "exclusivity failure".

In [3] Andersson describes a back door for an ATM system. The back door was a 14-digit number that forced ten banknotes to be paid out. The introduction of such a number, or rather the fact that it was not taken away during initial installation, constitutes a vulnerability. As a matter of fact, this number was documented in the maintenance manual (!), which was an error manifested in the documentation. The error propagated when a former maintenance engineer, in desperate need of money, recalled the number and started to make withdrawals from various ATM machines. This was the attack on the system. The fact that he could withdraw money is an exclusivity failure.

This example also clearly shows that faults may be introduced at very early stages in the system's life and may have a considerable latency period.

5. CONCLUSIONS AND FUTURE WORK

This paper makes a first attempt towards a unified terminology and understanding of security and dependability impairments. It has also been put into the context of an earlier work that addressed the same problem for concepts at a higher hierarchical level. The overall objective of this work is to arrive at a situation that would permit a unified and formal treatment of all aspects of the security/dependability meta-concept at the same time. This would pave the way for a holistic understanding of the problem area, which we think is necessary for the successful treatment of the existing problems.

It should be noted that this work simply represents another step towards a unified security/dependability framework and that it creates further questions. Obviously, reality is more complicated than is proposed in this simple binary model of intrusions and failures. We look forward to continuing the work towards a more comprehensive model that can also reflect that an intrusion may be a gradual penetration rather than an event-type phenomenon. See [23]. Similarly, real failures are better described as gradual degradations than as state changes [18]. Another interesting direction of this work is towards large distributed systems. We feel that finding a model for the definition and usage of impairments in these is definitely a non-trivial task.

6. REFERENCES

[1] E. N. Adams. Optimizing preventive service of software products. *IBM Journal of Research and Development*, 28(1): 2-14, 1984.

[2] P. Ammann, S. Jajodia. Computer Security, fault tolerance, and software assurance. *IEEE Concurrency*, Vol. 7, No. 1, January-March 1999.

[3] R. Andersson. Why cryptosystems fail. *Communications of the ACM*, 37(11), 1994.

[4] T. Anderson and P. A. Lee. Fault tolerance terminology proposals. In P. A. Lee and D. E. Morgan, editors, *Proceedings of the 12th IEEE International Symposium on Fault Tolerant Computing, FTCS-12*, pages 29-33, Santa Monica, CA, USA, June 1982.

[5] A. Avizienis. Fault tolerance, the survival attribute of digital systems. In *Proceedings of the IEEE*, 66(10):1109-1125, October 1978.

[6] A. Avizienis. The four-universe information system model for the study of fault-tolerance. In P. A. Lee and D. E. Morgan, editors, *Proceedings of the 12th IEEE International Symposium on Fault Tolerant Computing, FTCS-12*, pages 29-33, Santa Monica, CA, USA, June 1982.

[7] R. H. Baker. *Computer Security Handbook*, 2nd edition. TAB Professional and Reference Books, McGraw-Hill, 1991.

[8] M. Bishop and D. Bailey. A Critical Analysis of Vulnerability Taxonomies. Technical Report CSE-96-11, Department of Computer Science, University of California at Davis, CA, USA, September 1996.

[9] B. K. Daniels. Errors, faults and failures: A model. In T. Anderson, editor, *Safe and Secure Computing Systems*, Blackwell Scientific Publications 1989.

[10] D. E. Denning. Secure Databases and Safety: Some unexpected conflicts. In T. Anderson, editor, *Safe and Secure Computing Systems,* Blackwell Scientific Publications, 1989.

[11] H. Hedbom, S. Lindskog, E. Jonsson, " An Analysis of the Security of Windows NT", Tech. Rep. 99-16, Dept. of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1999.

[12] Institute of Electrical and Electronic Engineers. A Glossary of Software Engineering Terminology, Chapter 5, IEEE 610.12-1990.

[13] Information Technology Security Evaluation Criteria (ITSEC): Provisional Harmonized Criteria, December 1993.

[14] International Standards Organization. Data Processing: Open Systems Interconnection, Basic Reference Model, ISO/IS 7498, Geneva 1983.

[15] International Standards Organization. Information processing systems: Open Systems Interconnection, Basic Reference Model, part 2: Security Architecture 7498/2.

[16] E. Jonsson and T. Olovsson. On the Integration of Security and Dependability in Computer Systems. In *IASTED International Conference on Reliability, Quality Control and Risk Assessment*, Washington, USA, November 4-6, 1992.

[17] E. Jonsson. A Unified Approach to Dependability Impairments in Computer Systems. In *IASTED International Conference on Reliability, Quality Control and Risk Assessment*, pages 173-178, Cambridge, MA, USA, October 18-20 1993.

[18] E. Jonsson, M. Andersson, S. Asmussen, "A Practical Dependability Measure for Degradable Computer Systems with Non-exponential Degradation", In *Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, SAFEPROCESS'94,* Espoo, Finland, June 13-15, 1994, vol. 2, pp. 227-233.

[19] E. Jonsson, T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior", IEEE Transactions on Software Engineering, Vol. 23, No. 4, April 1997.

[20] E. Jonsson. An Integrated Framework for Security and Dependability. In *Proceedings of the New Security Paradigms Workshop 1998*, Charlottesville, VA, USA, September 22-25, 1998.

[21] I. V. Krsul. Software Vulnerability Analysis. *PhD thesis, Purdue University*, May 1998.

[22] J. C. Laprie et al. Dependability: Basic Concepts and Terminology. Springer-Verlag, 1992.

[23] U. Lindqvist, U. Gustafson, E. Jonsson, "Analysis of Selected Computer Security Intrusions: In Search of the Vulnerability", *NORDSEC'96 – Nordic Workshop on Secure Computer Systems,* Göteborg, Sweden, November 7–8, 1996.

[24] C. Meadows, Applying the Dependability Paradigm to Computer Security. In *Proceedings of the New Security Paradigms Workshop 1995*, La Jolla, CA, August 22-25 1995.

[25] C. Meadows, J. McLean, Security and Depandability: Then and Now. Presented at the *Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solutions*, Williamsburg, VA, November 11-13, 1998.

[26] D. M. Nessett. Factors Affecting Distributed System Security. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, pages 204-222, Oakland, CA, USA.

[27] National Institute of Standards and Technology. Glossary of computer security terms, NSC-TG-004 version. 1, ("Aqua Book"), October 21, 1988.

[28] S. M. Ornstein. Safety issues for computer controlled systems. In *Proceedings of the 16th IEEE International Symposium on Fault-Tolerant Computing, FTCS-16*, Vienna, Austriia, 1986.

[29] C. P. Pfleeger. Security in Computing. Prentice-Hall 1997. ISBN 0-13-185794-0.

[30] Department of Defence. Trusted Computer System Evaluation Criteria ("orange book"), CSC-STD-001-83.