The Remedy Dimension of Vulnerability Analysis

Ulf Lindqvist[1] Per Kaijser[2] Erland Jonsson[1]

[1]Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{ulfl, erland.jonsson}@ce.chalmers.se

[2]Siemens AG
DE-81730 München
Germany
Per.Kaijser@mchp.siemens.de

Abstract

This work is aimed at supporting system and information owners in their mission to apply a proper remedy when a security flaw is discovered during system operation. A broad analysis of the different aspects of flaw remediation has resulted in a structured taxonomy that will guide the system and information owners through the remedy identification process. The information produced in the process will help to make decisions about changes to the system or procedures. A selected vulnerability that was able to be removed using three different remedies is used as an example.

Keywords: Remedy, taxonomy, vulnerability, security, system owner, information owner.

1 Introduction

When the discovery of a security flaw in a system has come to the knowledge of a party that risks suffering a direct loss if the vulnerability were to be exploited by an attacker, that party must decide what remedial action to take in order to remove the flaw from the system. The party in question is usually the organization that owns the system and/or the information stored and processed in the system. The work presented in this paper is aimed at supporting the owners in their mission to apply a proper remedy once a flaw is discovered, by providing them with a framework for remedy identification and analysis.

The traditional and most common situation in larger organizations is that the ownership of, or right to access, data and information stored and processed in a system coincides with the ownership of the system[3]. For smaller organizations, particularly for the many small and medium enterprises (SMEs), there is an increased interest in outsourcing, that is, letting a professional service provider own, manage and operate "your" system. The result is that information owners and system owners belong to different organizations. Still, both types of owners are vulnerable to security breaches. An information owner risks losing control of the information, and a system owner may be forced to pay damages or risks losing customers through a bad reputation.

To improve the security of IT systems, several guidelines and standards have been produced. These have been aimed at the different actors that have an effect on the security of the IT system, such as the manufacturers, procurers, managers, operators and users. For vendors and manufacturers, the functionality of the system and the development processes have been the target of standards (TCSEC [18], ITSEC [6], CC [7]) that specify criteria against which security evaluations can be made. These have also led to an increased interest in research on formal methods [11, 1, 16]. For procurers, baseline security documents have been created that give a minimum set of requirements on security features that an IT system should possess [20]. Managers, operators and users of an IT system need to follow certain rules in the form of security policies in order to minimize potential threats. For this purpose, guidelines and codes of practice have been specified [4, 9].

In spite of all these efforts, the number of security vulnerabilities can only be reduced, not eliminated. But what is more important: Only a minor part of the systems trusted with valuable information in trade, industry, public services and academia today are designed and implemented according to these criteria. Further, the ways in which hardware and software can be combined and interconnected are so complex that not even experts can fully understand how to avoid vulnerabilities. Reports about new vulnerabilities in computing systems are issued on almost a daily basis, for example in CERT advisories (such as [5]) posted on the Internet. It is true that

[3]*Information owner* is normally referred to information with intellectual property rights (IPR). However, it can also be used to denote the individual or organization that is authorized to control access to a piece of information that might or might not be IPR protected. In the context of this paper, *information owner* covers both cases.

security policies and recommendations for system and information owners play an important role, but they will not solve all weaknesses. To put it briefly:

- Vulnerabilities exist and will remain in all systems in operation.

Everyone should realize that, whatever precautions are taken, security flaws will be present in systems when delivered and in operation and that we need to form strategies for dealing with these flaws. However, this does not mean that we propose a penetrate-and-patch doctrine; it is still very important to try to eliminate as many security flaws as possible during early phases of system development, because the costs and risks associated with a repair increase dramatically later in the product life cycle. We want to emphasize that security should be considered throughout the entire system life cycle and that the efforts in different phases complement one another.

It is the owners of the information and the owners of the system who are primarily exposed to security risks. Therefore, our work aims at supporting the owners in this situation. The authors hope that the results of this work will also be beneficial to the security community in a wide sense, including international industrial consortia such as I-4[4] and ESF[5], incident response teams such as CERT and, of course, system and information owners.

In the following, Section 2 describes some earlier work in the field, while our analysis of the remedy dimension and our proposed taxonomy are presented in Section 3. Examples of remedies follow in Section 4, and some conclusions are drawn in Section 5.

2 Previous work

In our previous work on categorization[6] of intrusions, we made some general observations on the design of categorization schemes [15]. First, it is important to clearly state the attribute or view of the intrusion on which the categorization was based. We suggested the use of the term *dimension* for that view. Second, it is desirable to have mutually exclusive and collectively exhaustive categories, but this is often difficult, or even impossible, to fulfil. Third, the true value of such a taxonomy is that its formation and application enforces a structured analysis, which clarifies the matter and can generate new ideas.

The choice of the remedy attribute as a further dimension for categorization of vulnerabilities is natural and significant. It encompasses the whole life cycle of an IT system and focuses on the parties exposed to the risks that the vulnerabilities represent. Until now, little has been published on how to actually perform remedy planning and analysis, although several authors have observed the need for such activities. In an insightful paper, Kahn and Abrams stressed the importance of anticipating system security failures and planning for recovery and remediation [10]. Risk management, flaw remediation and evolutionary development is argued to provide more cost-effective and up-to-date security assurance than the TCSEC model of risk avoidance and static systems.

In the Common Criteria [7], there is a so called assurance family named *Life cycle support—Flaw remediation* (ALC_FLR). Earlier drafts of the criteria were studied by van Laenen [19], who points out the problem of re-evaluation when a change to the system has been made and also suggests two new requirements to be considered: *Mean time to remediation* and *Maximum time to remediation*. It should be noted, however, that this concerns only remedies provided by the developer.

3 Remedy analysis and taxonomy

In the terminology used in the field of dependable and fault-tolerant computing systems [13], the term *fault prevention* is used for methods that prevent faults from occurring or being introduced into a system. Actions that aim to reduce the presence of faults that already have been introduced fall under the category of *fault removal* and, especially, fault removal encountered during the operational phase of a system's life is called *corrective maintenance*. The present paper investigates corrective maintenance applied to faults that cause security failures or, in other words, *the remedy dimension* of security vulnerabilities.

The remedy clearly depends on the nature of the vulnerability, but several new aspects must be carefully taken into account:

- What and who has caused the problem?
- Is there a possible way to remove the flaw?
- Will the changes introduce new vulnerabilities?
- Will the changes affect the quality of service?
- Will the changes actually remove the vulnerability?
- What will it cost to make the changes?
- What action should be taken—should any changes be made at all? If so, by whom?

[4] International Information Integrity Institute, a part of SRI Consulting which in turn is a subsidiary of SRI International. WWW: https://rome.isl.sri.com/i4/

[5] European Security Forum, Plumtree Court, London EC4A 4HT, England

[6] We prefer the term *categorization* to *classification*. The reason is that in the security field, the latter term is traditionally associated with a very specific dimension, namely, levels of confidentiality.

The system owner can become aware of a vulnerability through internal experience as well as from external sources such as a product developer who provides a correction, an alert group such as CERT which may point out a vulnerability or from its own customers (the information owners). The information owner—the organization to which all users of the system belong—normally comes to know about a fault after having had practical experience with it, but may also be informed of it by others, for example the system owner.

First, the source of the vulnerability should be identified. This includes identification of the technical location of the fault in the system as well as identification of the organizational unit whose activities introduced the fault. The latter may for example be the producer of the product, the system owner or the information owner.

The next step is to turn to those believed to be able to provide a solution to the vulnerability. This is preceded by an analysis of possible locations of a remedy. The owner then informs the potential remedy providers about the problem, its location in the system or the process that is believed to cause it and, possibly, gives suggestions for how to overcome it. The result may be one or more proposed remedy actions that must be carefully analyzed with respect to their impact on the system and on system operation and use.

Before and after each of these steps, a decision must be made as to how to proceed. Is it worth continuing the remedy process? And, if it is considered worthwhile to continue and there are alternatives, which one(s) should be taken? All these decisions must be based on facts and be viewed in the light of economic constraints. To support and aid the system and information owners in their decisions, a four-stage remedy identification process is proposed. The properties to be identified and analyzed are:

- Fault location
- Remedy location
- Remedy provider
- Remedy impact

Each of these stages are described in detail below.

3.1 Identification of fault location

The point in the system structure, operation or use at which the fault causing the vulnerability is located is called the fault location. Since a vulnerability might consist of a combination of circumstances [14], it may not be possible to distinctively identify a single point as the location of the fault. Still, the analysis is a necessary starting point in the remedy process.

Our taxonomy on fault location is shown in Table 1. The taxonomy of computer program security flaws presented by Landwehr *et al.* [12] partly serves the same purpose, and our categorization can be viewed as a combination and extension of the dimensions they call *location* and *time of introduction*. The reader is urged to note that, although the same diagram is used for categorization of the remedy location in the following subsection, the remedy location does not always coincide with the fault location (see the example in Section 4).

Fault location or Remedy location	Product/ solution	Requirements
		Design
		Implementation
	Integration	Requirements
		Design
		Implementation
	Installation	External issues
		Internal issues
	Operation/ administration	Policy
		Monitoring and enforcement of policy
		Instructions
	Use	Policy
		Monitoring and enforcement of policy
		Instructions

Table 1: Taxonomy of fault location or remedy location.

We will describe some of the categories below, hoping that the names of the other categories will be self-explanatory. We consider a fault to be located in the *integration* if a component is vulnerable as part of one system but not of another. In the *installation* category, *external issues* are located outside the chosen system boundary (for example, physical protection) while *internal issues* are initial configuration parameters etc.

The second-level categories below the top-level categories *operation/administration* and *use* may also call for some clarification. A *policy* (or, more specifically, a *security policy*) is basically a set of rules stating what is allowed, what is not allowed and what must be done. *Monitoring and enforcement of policy* concern the management's efforts to make certain that the policy is respected and obeyed (a flaw may consist in the lack of enforcement of an existing and appropriate policy). To help administrators and users to operate the system in a way consistent with the policy, *instructions* are required. For example, if the policy states that owners and users must take all reasonable action to prevent passwords from being revealed to an attacker, then the instructions should, for example, tell the system owner how to operate the system so that passwords are never sent in the clear via an untrusted network.

The result of this phase of the remedy identification process is the structural location of the cause of the fault. This will aid the owner in the next step of the process, namely, in determining where a remedy should be applied.

3.2 Identification of remedy location

Now the owner wishes to identify where in the system structure, operation or use a remedy should be applied. This step is based on the result of the fault location identification as well as on the type of flaw. The reason for this categorization is twofold: first, to be able to find the most appropriate remedy provider (see Section 3.3) and, second, to be able to estimate the remedy impact (see Section 3.4).

It should be noted that, from this stage and onward, several alternative proposed remedies to the same flaw may co-exist, each with its own location, provider and impact. The alternatives need not even be mutually exclusive, for example certain flaws might be of such a severe nature that an immediate remedy action is required until a more proper solution can be produced and applied. This stage in the process needs to be revisited as new proposals result from the owner's contacts with possible remedy providers.

The scheme for categorization of remedy location is identical to that of fault location, as shown in Table 1.

3.3 Identification of remedy provider

By remedy provider, we mean the party that needs to make the necessary changes in a component or process in order to remove the vulnerability. Identification of the remedy provider is naturally closely related to the remedy location, but is also related to the fault location. The organization behind the process in which the flaw was introduced is probably, but not necessarily, the best suited to provide information leading to a remedy. Furthermore, the originator of the fault has at least a moral, if not legal, responsibility to fix the problem.

The taxonomy of remedy provider is shown in Table 2. The categorization is based on the fact that the top-level categories often represent different organizations. *Product developers* are responsible for the production and maintenance (in terms of error correction and evolution) of the product. *Integrators* are responsible for the integration of products and solutions into a workable system. *Solution providers* are responsible for customer-specific solutions.

It should be noted that the *system* and *information owners* are also actively involved in the maintenance of the system, regardless of which party provides the actual remedy. Whereas a product developer often provides the technical solution for the removal of a fault, it is the system owner's administrators and operators that perform the update or ask the information owners to do it at their sites.

Remedy provider	Product developers	Designers
		Implementors/ maintenance
	Integrators	
	Solution providers	
	System owners	Policy makers
		Administrators/ operators
	Information owners	Policy makers
		Administrators/ operators
		End-users

Table 2: Taxonomy of remedy provider.

The identification of the remedy provider together with the result of the remedy location analysis helps the owner to identify the best way to remove the vulnerability. In the final step of the process, the results of the responses from the remedy providers (the suggested remedies) must be analyzed with respect to their impact on the system and the business.

3.4 Analysis of remedy impact

It is important to thoroughly analyze the impact of a suggested remedy on the system before applying it. If there are several different suggestions, the analysis could also help in choosing the optimum solution. This fourth step in the remedy process is based on our taxonomy of remedy impact as shown in Table 3. The reader should note that the category groups numbered 1 through 10 in Table 3 are not mutually exclusive. In fact, every remedy analyzed with respect to impact should be assigned to a single subcategory within each of these 10 different groups, as illustrated in Section 4.

The *primary technical effects* category concerns to what extent the remedy takes care of the vulnerability and what effects it has on the functionality of the component in which the changes were applied. If a particular instance of the vulnerability is removed but the basic flaw endures, we consider the vulnerability to be *partly eliminated*. On the other hand, a *provisionally eliminated* vulnerability means that the flaw itself is not repaired, but the situation in which it can be exploited is rendered impossible, for the time being (a typical example is the shutdown of a faulty service). The category of *secondary technical effects* concerns whether a *new vulnerability* is

Category				No.
Remedy impact	Primary technical effects	Target vulnerability eliminated	Provisionally	1
			Partly	
			Completely	
		Functionality (of changed parts)	Impaired	2
			Unchanged	
			Improved	
	Secondary technical effects	Severity of identified new vulnerability	High	3
			Medium	
			Low	
			None	
		Functionality (of unchanged parts)	Impaired	4
			Unchanged	
			Improved	
	Primary economic effects (related to performing the change)	Cost for internal resources	High	5
			Medium	
			Low	
		Cost for external resources	High	6
			Medium	
			Low	
			None	
	Secondary economic effects (after the change)	Cost in human resources	Increased	7
			Unchanged	
			Decreased	
		Processing time	Increased	8
			Unchanged	
			Decreased	
		Cost in computer resources	Increased	9
			Unchanged	
			Decreased	
	Time to remediation		Long	10
			Medium	
			Short	

Table 3: Taxonomy of remedy impact.

introduced, and how the *functionality* of unchanged (but dependent) parts is affected.

In addition to technical effects, there are also economic effects of a remedial action. The only such effects of interest to the decision-maker are of course the ones concerning their own organization. We have also separated the economic effects into primary and secondary, where the former are related to the immediate cost of performing the change, while the latter concern the long-term consequences after the change is made. The immediate cost consists of the internal cost from workload among the owner's own staff and the cost of paying for equipment, services or solutions from external sources (depending on the contract situation). An example of long-term impact is the case in which the change results in a certain additional working time for some administrator or user tasks. On the other hand, if the system services are faster and simpler after the change, a secondary economic effect would be decreased processing time.

The time the owner needs to wait for the remedy is of course a significant factor in the decision process, since the system and information are vulnerable until the remedy has been applied. The category of *time to remediation* with the rough subcategories *long*, *medium* and *short* is meant to be used in a relative rather than absolute sense.

4 A vulnerability and examples of remedies

In this section, we present a well-known vulnerability and some examples of remedies in order to illustrate and exemplify the taxonomy presented in Section 3. For the sake of brevity, the technical description of the flaw is here kept to a minimum; the interested reader will find more detailed descriptions in the references cited.

4.1 The Unix X terminal emulator logging vulnerability

The X Window System terminal program *xterm*, running with the effective user id of *root* (super-user) in some Unix variants, had a flawed logging facility that could be used to create an arbitrary new file or modify any existing file by appending an arbitrary set of data to it [5, 14, 2].

Our first step is to identify the vulnerability. It turns out that, in the procedure implementing the logging facility, *xterm* makes certain critical system calls with the privileges of *root* instead of with the privileges of the user invoking the program. In this case, the flaw must clearly be categorized as being located in the product design, since the program presumes *root* privileges but is not designed with the precaution needed for privileged programs.

A correction can either be provided by the product developers, product maintenance or even the system owner, giving rise to three alternative remedy actions for us to consider:

a) Remove the super-user privileges from *xterm*

b) Disable the logging facility of *xterm*

c) Rewrite *xterm* according to “the principle of least privilege”

We thus note that there is only one entry for fault location whereas there are three for both remedy location and remedy provider.

4.2 Remedy a: remove the super-user privileges

The quickest and easiest way to remedy the *xterm* flaw is to clear the set-user-id flag of the program, that is, to remove its super-user privileges. However, there is a reason why *xterm* was installed with those privileges: it needs to change the owner of the pseudo-terminal slave device, an action which requires *root* access[7]. When testing this remedy on a SunOS 4.1.2 system in a university computer security laboratory, we found the following:

- To the terminal user, *xterm* appears to function normally.
- The pseudo-terminal slave device, to which *xterm* connects, is still owned by *root*. In order for the device to be readable and writable for the user, it must be so for a group of users or even all users. Consequently, a new vulnerability is introduced, primarily threatening the user rather than the system owner.
- If the system logging file */etc/utmp* is writable only to *root* (which was the case in the test system, since a writable */etc/utmp* constitutes another vulnerability, see [14]) the terminal connection is not reported by some programs that list the users logged on to the system, for example *who*.

We can only consider the vulnerability to be provisionally eliminated by this remedy because, for example, an uninformed administrator who discovers the device and logging problems might assume that the privileges have been turned off by mistake or by accident. If the administrator turns the privileges back on, the system would again be vulnerable. Any economic effects of this simple remedy are negligible.

With the information at hand, we make the following categorization of this remedy:

Fault location: Product: Design

Remedy location: Installation: Internal issues

Remedy provider: System owners: Administrators/operators

Remedy impact: see Table 4.

4.3 Remedy b: disable the logging facility

The patches distributed by the developers in response to the CERT warning disable the logging facility. The function of the logging facility is to provide a simple means for the user to save the terminal output in a file, a service evidently impaired by this remedy. Another disadvantage of this approach is that it does not solve the basic problem, namely, that *xterm* unnecessarily makes all its system calls with *root* privileges, leaving the system vulnerable to a number of other, similar and yet undiscovered flaws (see the discussion on Remedy c below).

Fault location: Product: Design

Remedy location: Product: Implementation

[7] There are Unix variants in which *xterm* works without special privileges. In such cases, this particular vulnerability never existed, and the remedy discussion is a non-issue.

Impact of remedy		a	b	c	No.
Primary technical effects	Target vulnerability eliminated	Provisionally	Partly	Completely	1
	Functionality	Unchanged	Impaired	Unchanged	2
Secondary technical effects	Severity of identified new vuln.	Medium	None	None	3
	Functionality	Impaired	Unchanged	Unchanged	4
Primary economic effects	Cost for internal resources	Low	Low	Low	5
	Cost for external resources	None	None	*Depends*	6
Secondary economic effects	Cost in human resources	Unchanged	Unchanged	Unchanged	7
	Processing time	Unchanged	Unchanged	Unchanged	8
	Cost in computer resources	Unchanged	Unchanged	Unchanged	9
Time to remediation		Short	Medium	Long	10

Table 4: Impact analysis of the three suggested remedies.

Remedy provider: Product developers: Implementors/maintenance

Remedy impact: see Table 4.

4.4 Remedy c: rewrite the program

Regardless of whether the concern is reliability, safety or security, it has long been known that critical regions of software should be as small and simple as possible, since complex programs are error-prone. It is also well-known in the security community that no action should be performed with higher privileges than those absolutely necessary to complete the task. These two rules are known as the principles of "economy of mechanism" and "least privilege", respectively [17]. The flawed version of *xterm* is a large program running with constant super-user privileges, although such powers are necessary only for a small fraction of its duties. Unfortunately, *xterm* is not the only Unix utility that violates both of the above principles; *sendmail* is another notorious example.

This remedy action suggests *xterm* to be rewritten in a defensive, security-conscious programming style, following "best practice" guidelines for privileged programs [3, 8]. In this case, not only the logging flaw would be eliminated, but also any similar security flaws in other parts of the code. This is a relatively expensive solution, however, and there is always a risk that new security flaws and other bugs are introduced when such a large piece of software is modified extensively.

Fault location: Product: Design

Remedy location: Product: Design

Remedy provider: Product developers: Designers

Remedy impact: see Table 4.

4.5 Discussion

In our example, a single fault was able to be removed using three different remedies with different impacts. Thanks to the taxonomy, the three remedies can be easily compared. It gives the system owner well-founded facts for making a decision, and we see that even if remedy c is preferred, either a or b could probably be accepted as a temporary solution.

How different owners would categorize a certain aspect of a given remedy may appear somewhat subjective. It should be remembered that each owner needs to find and validate remedies according to site-specific circumstances. Categorizations might therefore vary for different owners. Future development of categorization criteria could perhaps further help users of the taxonomy.

5 Conclusions

The purpose of the remedy identification process defined and described in this paper is to aid and support those exposed to the threats—the system and information owners—in how to proceed in their decision process, rather than to design a remedy for a given vulnerability. Our process consists of four phases; locating the fault, locating the remedy, identifying the provider of the remedy and analyzing the impact of the remedy. Each of these phases requires a taxonomy for easy categorization. The paper has described four taxonomies that are the core of the analysis in each of these steps. The remedy location and remedy provider phases can be iterative, since the result of each may require an update of the other. This is different from the first phase (fault location), which only serves as initial input, and the last phase (remedy impact), which analyzes the output from the two middle phases. The final result—the impact of each proposed remedy—is the desired outcome on which the owner can base a sound decision as to how to proceed.

References

[1] M. D. Abrams and M. V. Zelkowitz. Striving for correctness. *Computers & Security*, 14(8):719–738, 1995.

[2] T. Aslam, I. Krsul, and E. H. Spafford. Use of a taxonomy of security faults. In *Proceedings of the 19th National Information Systems Security Conference*, pages 551–560, Baltimore, Maryland, Oct. 22–25, 1996. National Institute of Standards and Technology/National Computer Security Center.

[3] M. Bishop. How to write a setuid program. *;login: (The USENIX Association Newsletter)*, 12(1):5–11, Jan./Feb. 1987.

[4] British Standards Institution. *Code of Practice for Information Security Management*, 1995. BS 7799.

[5] CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA. *xterm Logging Vulnerability*, Nov. 11, 1993. CERT Advisory CA-93:17.

[6] Commission of the European Communities. *Information Technology Security Evaluation Criteria*, June 1991. Version 1.2.

[7] Common Criteria Implementation Board. *Common Criteria for Information Technology Security Evaluation*, May 1998. Version 2.0. See also ISO/IEC 15408.

[8] S. Garfinkel and G. Spafford. *Practical UNIX & Internet Security*. O'Reilly & Associates, second edition, 1996.

[9] INFOSEC Business Advisory Group. *The IBAG Framework for Commercial IT Security*, Sept. 1993. Version 2.0.

[10] J. J. Kahn and M. D. Abrams. Contingency planning: What to do when bad things happen to good systems. In *Proceedings of the 18th National Information Systems Security Conference*, pages 470–479, Baltimore, Maryland, Oct. 10–13, 1995. National Institute of Standards and Technology/National Computer Security Center.

[11] C. E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, Sept. 1981.

[12] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3):211–254, Sept. 1994.

[13] J.-C. Laprie, editor. *Dependability: Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, 1992.

[14] U. Lindqvist, U. Gustafson, and E. Jonsson. Analysis of selected computer security intrusions: In search of the vulnerability. Technical Report 275, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1996. Presented at NORDSEC – Nordic Workshop on Secure Computer Systems, Göteborg, Sweden, Nov. 7–8, 1996.

[15] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 154–163, Oakland, California, May 4–7, 1997. IEEE Computer Society Press, Los Alamitos, California.

[16] P. G. Neumann. Architectures and formal representations for secure systems. Technical Report SRI-CSL-96-05, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, May 1996.

[17] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept. 1975.

[18] U.S. Department of Defense. *Trusted Computer System Evaluation Criteria*, Dec. 1985. DoD 5200.28-STD.

[19] F. van Laenen. Pedigree and credentials, remediation and legal aspects to gain assurance in IT products and systems. Master's thesis, Katholieke Universiteit Leuven, Belgium, and Norges Tekniske Høyskole, Norway, 1995.

[20] X/Open Company Ltd., UK. *X/Open CAE Specification: Baseline Security Services (XBSS)*, 1995. X/Open Document Number C529.