Observations on Operating System Security Vulnerabilities

STEFAN LINDSKOG

Department of Computer Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2000
Observations on Operating System Security Vulnerabilities

STEFAN LINDSKOG

Department of Computer Engineering
Chalmers University of Technology
Göteborg, Sweden 2000
Observations on Operating System Security Vulnerabilities

STEFAN LINDSKOG
Department of Computer Engineering
Chalmers University of Technology

Thesis for the degree of Licentiate of Engineering

Abstract

This thesis presents research on computer security vulnerabilities in general-purpose operating systems. The objective is to investigate intrusions in such systems in order to find and model the underlying generic weaknesses, i.e., weaknesses that would be applicable to many different systems. An attempt is made to create a conceptual basis for the generic modeling of vulnerabilities, addressing security concepts, definitions, and terminology. The investigation of intrusions is based on empirical data collected from three different systems, UNIX, Novell NetWare, and Windows NT. The UNIX and Novell NetWare data were generated from a number of practical intrusion experiments with Master’s students, while the Windows NT data resulted from a security analysis that we performed ourselves. This analysis showed that Windows NT, initially thought to be quite secure, still displayed a significant number of vulnerabilities. A comparison with earlier UNIX analyses indicates that the security differences between the systems are related more to factors such as time on market and security-by-obscurity than to inherent security performance. A different approach was taken with the Novell NetWare system. Here, the initial experiments showed quite poor security behaviour and we have investigated the security evolution as later versions of the system have been released and have reached the conclusion that, even though some effort has been made to improve security, no significant difference has been achieved, primarily due to backward compatibility requirements. In summary, the work performed here represents a further step towards a full understanding of the generic weaknesses and vulnerabilities that impair commercially available operating systems. This understanding is essential to our aspiration to make these systems secure, or at least sufficiently secure. It is expected that this work, when fully accomplished, will comprise a powerful basis for improving the security of operating systems, at least to the extent that research results are taken into consideration by software developers and manufacturers.

Keywords: computer security, operating system, vulnerability, intrusion, experiment, modeling, analysis.
List of Publications

This thesis is based on the work presented in the following papers:

I. Security Basics – Definitions and Input Data


II. Security Analysis


# Contents

Abstract ................................................................. iii

List of Publications ......................................................... v

Introductory Summary ......................................................... 1

1 Introduction ............................................................... 3

2 On Operating System Security ............................................. 4

3 Security Evaluation ........................................................ 5

4 Operating System Vulnerabilities .......................................... 6
   4.1 Terminology ............................................................ 6
   4.2 Categorization of Security Vulnerabilities ......................... 7
   4.3 Observations on Common Security Vulnerabilities ............... 9

5 Summary of Papers ........................................................ 13
   5.1 Part I: Security Basics – Definitions and Input Data ........... 13
   5.2 Part II: Security Analysis ............................................. 14

6 Discussion ................................................................. 15

7 Results .......................................................... 16

Acknowledgments ............................................................ 17

References ................................................................. 18

Part I ................................................................. 23

Paper A ................................................................. 25

Paper B ................................................................. 45

Part II ................................................................. 63

Paper C ................................................................. 65

Paper D ................................................................. 91

Paper E ................................................................. 111
Introductory Summary
1 Introduction

Modern organizations depend heavily on their information systems and large investments are made in them annually. These systems are now computerized, and connecting computers to networks has been the common trend during the last decade. The availability of information and computer resources within an organization as well as between cooperating organizations is often critical for the production of goods and services. In addition, data stored in or derived from the system must be correct, i.e., data integrity must be ensured. In some situations, it is also of great importance to keep information confidential.

Computer security is traditionally defined by the three attributes of confidentiality, integrity, and availability [17, 33]. Confidentiality is the prevention of unauthorized disclosure of information. Integrity is the prevention of unauthorized modification of information, and availability is the prevention of unauthorized withholding of information or resources. Other definitions of computer security have been suggested as well. Recently, Jonsson [20] proposed a novel approach to the integration of security and dependability. In that paper, he showed that there is a considerable conceptual overlap between the two disciplines. In [Paper A], we also show that there is a conceptual overlap of impairments from these areas and suggest an integrated approach that clarifies the functional relation between them.

This thesis focuses mainly on security vulnerabilities in general-purpose operating systems. The objective is to investigate intrusions in such systems for the purpose of searching for generic weaknesses. The thesis is based on empirical data primarily collected from three different systems, UNIX\(^1\), Novell NetWare, and Windows NT. [Paper B] provides a survey of the student intrusion experiments conducted at the Department of Computer Engineering, Chalmers University of Technology. A security analysis of Windows NT is presented in [Paper C], while security features and security vulnerabilities in Windows NT and UNIX are compared in [Paper D]. The security evolution of Novell NetWare is covered in [Paper E].

In the following, Section 2 gives a brief introduction to operating system security. Computer security evaluation methods are presented in Section 3. Section 4 covers operating system vulnerabilities based primarily on our own security analyses. A summary of the appended papers is given in Section 5. Section 6 discusses the underlying reasons for the vulnerabilities addressed in the thesis and, finally, the results are summarized in Section 7.

---

\(^1\) When using the term UNIX, we refer primarily to SunOS versions 4.1.2 and 4.1.3 from Sun Microsystems.
2 On Operating System Security

An operating system is a fundamental component of most modern computer systems. This component can be viewed in many different ways, since it is difficult to make a precise definition of the term. One common view is that it is a virtual or extended machine [40, 43, 44]—it masks the details of the underlying hardware from the programmer and provides the programmer with a convenient interface for using the system. On the other hand, an operating system can be viewed as a resource manager, responsible for fair resource sharing between different processes in the system. When multiple concurrent users are active within a system, the operating system must also ensure protection of the computation of one user from another.

The basis for protection is separation. Rushby and Randell [35] stated that separation in operating systems, or rather distributed systems, can occur in four different ways: (1) physical separation, (2) temporal separation, (3) logical separation, and (4) cryptographic separation. Combinations of two or more of these are of course also possible. In MULTICS [39], for example, run-time protection is implemented through the file system and a ring structure. An access control list (ACL) is associated with each file. This is roughly the protection facility offered by the file system. In addition, a process operates within a so-called protection domain, which specifies the resources the process may access. Protection domains are organized into a ring structure, and each ring corresponds to a single domain. The rings are numbered from 0 and up, the kernel being ring 0. The ring to which a process belongs determines its privileges—the lower the ring number the more the privileges.

The MULTICS system was in its entirety a very ambitiously designed operating system at the time, and that holds even when it is compared with current general-purpose operating systems. The security features offered by today’s systems often include memory protection, file protection, object (or resource) protection, access control to objects, and user authentication. In addition, different execution modes are used to protect (or separate) the operating system from user processes. However, only two different execution modes are normally used, user mode and kernel mode, even though the underlying hardware has built-in support for more. The operating system kernel executes in kernel mode, while programs initiated by users execute in user mode. In kernel mode, no access restrictions are enforced.

However, the kernel is not the only component in modern operating systems. Much system functionality is implemented outside the kernel through user level processes with extended privileges. These so-called server or daemon processes are most often initiated during bootstrapping. Vulnerabilities in these processes can be almost as serious as vulnerabilities in the kernel, due to the fact that they are executing with such powerful privileges. Indeed, these server processes seem to be more vulnerable than the kernel, as very
few kernel vulnerabilities have been exploited in our security experiments, see for example [Paper C, Paper D, Paper E].

3 Security Evaluation

Computer systems and especially their protection mechanisms must be penetration resistant. However, most, or perhaps all, current systems have security holes which make them vulnerable. As long as vulnerabilities remain in a system, the security may be circumvented. It is thus not difficult to understand why system owners would like to know how secure their particular system actually is.

Security evaluations have been performed for quite some time by mainly two different methods. The first is to classify systems into a predefined set of categories, each with different security requirements. The Trusted Computer Security Evaluation Criteria (TCSEC) [36, 37, 45] proposed by the Department of Defense of the United States (US) and the Information Technology Security Evaluation Criteria (ITSEC) [17] suggested by the European Union (EU) are two well-known examples of such classification schemes. It is notable that the US, Canada, and EU have recently developed the Common Criteria for Information Technology Security Evaluation (CC) [5]. Version 2.1 of the CC is the most recent and has been adopted by ISO as an international standard, ISO/IEC 15408:1999.

The other method is referred to as penetration testing [2, 8, 12], which is a form of stress testing exposing weaknesses in a system. Here, the idea is to let a group of people, normally very skilled in the computer security area, attack a target system. The earliest described and used method for conducting such testing is the Flaw Hypothesis Methodology (FHM) [26, 46]. The FHM uses a white-box testing [41] strategy, since both target system expertise and source code must be available to the penetration team.

Risk analysis may also be regarded as a security evaluation method. Here, the risk associated with the usage of a computer system in a particular way and in a particular configuration is determined. The basic steps in a risk analysis, as stated by Pfleeger [33], are: (1) identify assets, (2) determine vulnerabilities, (3) estimate likelihood of exploitation, (4) compare expected annual loss, (5) survey applicable controls and their costs, and (6) project annual savings of control. However, risk analysis is most often used in conjunction with security planning in order to describe the current security status. The drawback of risk analysis is that it is based primarily on subjective estimation.

Intrusion experiments have been conducted at the Department of Computer Engineering at Chalmers University of Technology since 1993. A black-box testing [41] strategy

---

1 Bootstrapping, often abbreviated booting, is the process of starting up a computer from a halted or powered-down state.
was used, since neither system expertise nor source code were available to the attackers. In all, three commercial, general-purpose operating systems, SunOS [4, 11, 21, 31, 32], Novell NetWare [9, 10, Paper E], and Windows NT [Paper C, Paper D], and a secure database system [29] have been studied. The security of SunOS and Novell NetWare was evaluated by letting final year Master of Science students in Computer Science and Engineering attack the systems. The original goal of these experiments was to find a quantitative measure of operational security [4, 9, 31, 32]. For this reason, the attackers were selected as far as possible to resemble “normal users”, rather than using persons with certain skills and knowledge. A summary of the experiments is presented in [Paper B]. Furthermore, procedures inspired by the intrusion experiments were used in an analysis of Windows NT [Paper C, Paper D], and in an analysis of a secure database system [29]. In those cases, we adopted the role of attackers, presumably more like tiger team attackers [2, 8, 12], ourselves.

4 Operating System Vulnerabilities

Vulnerabilities will probably always exist in large and complex software systems. At least with today’s software methods, techniques, and tools, it seems to be impossible to completely eliminate all flaws. Operating systems are examples of software components that are large, very complex, and vulnerable. At the same time, these components play an important role in the achievement of overall system security, since many protection mechanisms and facilities, such as authentication and access control, are provided by the operating system.

4.1 Terminology

In order to avoid misunderstandings, a few essential terms are defined in this subsection. The terms introduced are attack, breach, intrusion, vulnerability, flaw, weakness, fault, error, and failure. It should be noted that neither undisputed nor established definitions exist. However, the definitions given below are believed to have widespread acceptance and are used throughout this thesis, unless otherwise specified. More formal definitions of the terms are given in [Paper A].

An attack is an intentional activity conducted or initiated by a human, attempting to cause a breach in a system or to compromise the system. A breach is the resulting violation of the security policy of a system. An attack that results in a breach is referred to as a successful attack. We use the term intrusion (or penetration) to denote an attack and its corresponding breach.
A vulnerability is defined as a place at which the probability of a breach exceeds a predefined threshold. In this context, vulnerability, flaw, and weakness are regarded as synonyms, and will be used interchangeably. Note that we have intentionally avoided the terms fault, error, and failure, because they are used with different meanings in different communities. In the fault-tolerance community [25], for example, a fault is the cause of an error, while the software engineering community [18] claims that a fault is the result of a programmer’s error. See [22] for an in-depth discussion of this topic.

Similar definitions of security terms have been suggested. Howard and Longstaff [16], for example, recently proposed a “common language” of computer security terms, which consists, besides terms, of a taxonomy used to classify computer and network security incidents. Key words in the taxonomy are incident, attackers, attack, tool, vulnerability, action, target, event, unauthorized result, and objectives. To a large extent, their definitions agree with the definitions given above.

4.2 Categorization of Security Vulnerabilities

By categorizing phenomena, it becomes much easier to conduct systematic studies. Several categorization schemes for security vulnerabilities have been proposed through the years. A survey of previous work is found in [28]. In this section, we emphasize only three recently published taxonomies [1, 23, 24]. A summary of these (in chronological order) is given below.

Landwehr et al. [24]. Landwehr et al. stated in the introduction of their article that:

“Knowing how systems have failed can help us build systems that resist failure.”

They also observed that the history of software failures is mostly undocumented. These two observations motivated them to develop a taxonomy of computer program security flaws, which is largely based on vulnerabilities in operating systems. In the paper, 50 security flaws, described previously in the open literature, are studied and categorized. The vulnerabilities were not randomly selected, owing to the fact they were not taken from a valid statistical sample, but were rather selected to represent a wide variety of security flaws. The taxonomy consists of three general categories: genesis, time of introduction, and location. These are sometimes referred to as dimensions, and a taxonomy with more than one dimension is called a matrix [16].

- Genesis answers the question “How did the vulnerability enter the system?”
- Time of introduction answers the question “When did the vulnerability enter the system?”
- Location answers the question “Where in the system was the vulnerability introduced or found?”
Table 1. The genesis dimension of security flaws [24]

<table>
<thead>
<tr>
<th>Intentional</th>
<th>Malicious</th>
<th>Non-replicating</th>
<th>Replicating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trojan horse</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trapdoor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logic/Time Bomb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonmalicious</td>
<td>Covert channel</td>
<td>Storage</td>
<td>Timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Inadvertent | Validation error   |                 |             |
|            | Domain error       |                 |             |
|            | Serialization/aliasing |             |             |
|            | Identification/authentication inadequate |             |             |
|            | Boundary condition violation |             |             |
|            | Other exploitable logic error |             |             |

These categories are divided into subcategories, which in turn are most often further subdivided. For example, the genesis dimension, see Table 1, consists of two subcategories, intentional and inadvertent. An inadvertent vulnerability is categorized into one of the following six subclasses: validation error, domain error, serialization/aliasing, identification/authentication inadequate, boundary condition violation, and other exploitable logical errors. A vulnerability will appear at least once in each of the three dimensions, and sometimes more than once. The fact that a given vulnerability may appear in more than one category has been a common source of criticism of the taxonomy, see for example [1, 16].

Aslam [1]. Aslam presented a taxonomy of security faults in the UNIX operating system as part of his Master’s thesis. The thesis also presents the design and implementation of a vulnerability database organized according to the taxonomy. The terms fault, error, and failure are used as suggested in the standard glossary of software engineering terminology [18], where a fault is the result of an error caused by a human and a fault may lead to a failure. The taxonomy was derived from 49 documented security faults in the UNIX operating system. Data on faults were collected from different sources, including flaws reported on security mailing lists and Computer Emergency Response Team (CERT) advisories. An overview of the taxonomy is given in Table 2.

Operational, environment, and coding faults are the three general categories in Aslam’s taxonomy. Operational faults are caused by configuration mistakes, e.g., an object installed with incorrect permissions. Interaction errors between functionally correct components are typical examples of environment faults. The last category, coding faults, is caused either by condition validation errors, such as input validation errors, or by synchronization errors, e.g., race condition errors.
Krsul [23]. Extensions and revisions of Aslam’s categorization scheme [1] were proposed by Krsul as part of his doctoral dissertation “Software Vulnerability Analysis”. Two major extensions were proposed in the scheme. First, at each level of the taxonomy a complementary so-called “catch-all” category was introduced, e.g., at the top-most level an other fault category was added. Second, three new subcategories, excluding the “catch-all” category, of coding faults were included, authentication, program design error, and atomicity. A number of minor changes, such as renaming of categories and subcategories, were suggested as well.

### 4.3 Observations on Common Security Vulnerabilities

Observations on common operating system vulnerabilities, which are primarily derived from data collected in our security analysis summarized in Section 3 and described in [Paper B, Paper C, Paper D, Paper E], are presented in this section. Our ambition is not to suggest yet another vulnerability taxonomy but rather to emphasize common weaknesses that have been exploited in real intrusions. Similar vulnerabilities have been grouped together in a heuristic manner rather than according to some formal model. A collection of five common security problems has been identified and will be further described below. These are:

- Improper input validation
- Weak cryptographic algorithms
- Weak authentication protocols
- Insecure bootstrapping
- Configuration mistakes

The first four of these have a technical or system-related basis, while the latter is related to organizational problems or management. This implies that they are not orthogonal, i.e., a
specific vulnerability may belong to more than one group. Apart from describing the security problems, we also put them in relation to the taxonomies cited in Section 4.2. A few concrete examples illustrating the vulnerabilities are also given.

Not all vulnerabilities reported in the appended papers originate from the operating system itself. A badly chosen password, for example, is not an operating system vulnerability, but rather a user related weakness, caused by the fact that the user is either unaware of the possible consequences or just careless.

**Improper input validation.** In general, it is essential to carefully check the input to software routines, i.e., to perform an input validation. The check may be with regard to the number of parameters provided, the type of each parameter, or to simply ensure that the amount of input data is not larger than the buffer allocated to store the data. Improper or non-existent input validation is a well-known and serious problem in operating systems. Both Landwehr et al. and Aslam address these kinds of flaws in their taxonomies.

*Example 1.* Probably the best known example of an input validation vulnerability is the buffer overflow exploited in the Internet Worm [6, 34, 42]. In this case, the `fingerd` program was attacked. However, the vulnerability exploited was located in the `gets` routine in the standard C language I/O library\(^1\), which did not perform any validation at all of input bounds.

*Example 2.* Parameter validation in system functions is especially important, since these execute in kernel mode. However, not all operating systems perform proper control of system call parameters, e.g., Windows NT. This is demonstrated in the `NTCrash` utility, which is described in [Paper C].

*Example 3.* Buffer overflows in protocol implementations have also been very common in many of today’s network based operating systems. During the last couple of years a large number of exploit scripts demonstrating packet handling flaws have been published on the Internet. `Teardrop`, `bonk`, `boink`, `Land`, and `LaTierra`, see [Paper C, Paper D], are all examples of such scripts. Apparently, many router products on the market were also sensitive to these attacks.

**Weak cryptographic algorithms.** Overly weak algorithms in cryptographic systems are, in our opinion, another security problem. None of the classification schemes presented and described in Section 4.2 regard this as a security flaw. In operating systems, cryptographic algorithms have long been used to encrypt passwords. However, if the algorithm in use is not sufficiently strong, an attacker may manage to derive plaintext passwords from its corresponding encrypted representation. Note that a strong cryptographic algo-

\(^1\) Spafford [42] noted that a few other routines in the standard library have the same vulnerability.
rithm can suddenly become weak as a result of a research break through in, for example, number theory.

Example 4. In Windows NT, two hashed variants of each password are managed, one in Lan Manager (LM) and one in NT native format [Paper C]. The Lan Manager password is padded with NULLs if it is less than 14 characters long. This together with the encryption method used creates recognizable patterns if the password contains less than eight characters [15], which dramatically reduces the number of trials necessary in a brute force attack.

Example 5. Encrypted passwords in standard UNIX were stored in the public readable file, /etc/passwd. In the first password algorithm, the password was used as the key, and a constant was encrypted using this key [30]. The result from the algorithm is the encrypted password. However, if more than one user have selected the same password, they will also have exactly the same encrypted password stored in /etc/passwd, which might lead to accidental discovery of the other users’s password. This was not satisfactory. The solution to this problem, suggested by Morris and Thompson [30], was to introduce something that was called “salt”. The salt is a 12-bit random number, obtained from the real-time clock when a password is first entered. This random number is appended to the typed password before it is encrypted. The result from this encryption together with the salt is then stored in /etc/passwd. Hence, there are 4096 ($2^{12}$) encrypted versions of each password. This does not increase the amount of time required to search for a single user’s password, but it becomes more or less unfeasible to prepare an encrypted dictionary in advance, thus preventing parallel searches of many passwords simultaneously. An in-depth presentation of the UNIX encrypted password system is found in [7].

Weak authentication protocols. Before a user gains permissions to access resources, he must prove his identity. This process is called authentication. Most authentication systems are based on a secret common to the parties involved. The authentication mechanism is most often simply a password, i.e., a secret word known to the system and the user only. However, accomplishing a secure authentication procedure is a complex task, especially in a distributed environment.

Landwehr et al. [24] address this type of vulnerability as a subcategory called identification/authentication inadequate in the genesis dimension, and a subcategory called identification/authentication in the location dimension. Aslam, on the other hand, does not cover it, although the extended and revised version suggested by Krsul has a subcategory called authentication. Below, we illustrate weak authentication protocols with three man-in-the-middle attacks. They are thoroughly described in [Paper D].

---

1 “He” should be read as “he or she” throughout this thesis.
Examples 6 and 7. Two of the attacks are indeed very similar. The first is directed towards a remote logon in Windows NT, while the other is on the Network Information System (NIS). The latter is often used in a networked UNIX environment to facilitate centralized administration of the system. In both cases, the attacker manages to fool a victim client by masquerading as the server. Both attacks succeeded because of misplaced trust, i.e., the clients blindly trust the information without verifying that the server is the actual sender. A detailed description of these attacks is presented in [3] and [13].

Example 8. Windows NT supports eight different authentication variants with different security levels. The reason for this is backward compatibility. The client and the server negotiate over which authentication method should be used, see [Paper D]. Since the server will by default accept plaintext passwords as a valid authentication variant [14], it is sometimes possible for an attacker to fool a client into using a weaker authentication variant than necessary by interfering with the dialogue between the negotiating parties and to intercept the password later while it is being transmitted. None of the parties will know that such a downgrading attack was performed.

Insecure bootstrapping. From our security analyses it is apparent that system initialization (i.e., bootstrapping) is a major security problem in today’s operating systems. All studied systems were vulnerable during bootstrapping. Landwehr et al. use a subcategory called system initialization in the location dimension, while Aslam regards system initialization problems as an environment fault.

Example 9. In the first full-scale experiment [Paper B], many attackers discovered that the SunOS was easily rebooted in single-user mode. Commands entered in that mode run with root privileges [11], and it was possible to extend these privileges to the server, i.e., the whole network.

Example 10. A Windows NT system executing on a PC can most often be rebooted with a foreign operating system, such as MS-DOS. Once a foreign system is booted on the PC, the NTFS volume may be mounted. Access to files on the newly mounted volume will bypass the access control mechanism enforced when Windows NT is operating, see [Paper C].

Configuration mistakes. In current operating systems, security features and mechanisms are seldom activated by default. A secure “out-of-the-box” installation is the exception rather than the rule. To achieve an acceptable security level, the system owner must, after he has completed installation, spend quite some time in securing the system. However, securing a system is not a trivial task, since operating systems are large and complex. In addition, the number of skilled practitioners in computer security is very small [19]. In this thesis, vulnerabilities introduced during installation and/or configuration are collectively referred to as configuration mistakes. However, Aslam divided configuration flaws into
Observations on Operating System Security Vulnerabilities

three subcategories, object installed with incorrect permissions, utility installed in the wrong place, and utility installed with incorrect setup parameters. In the taxonomy proposed by Landwehr et al., configuration mistakes are covered in the time of introduction dimension as a subcategory, i.e., *during maintenance.*

*Example 11.* In the first SunOS experiment, see [11, Paper B], the NIS server configuration database directory `/var/yp` had permission mode `0777`, i.e.,

```
  drwxrwxrwx  5  root  staff  Sep 30 1992  512  yp
```

This implies that the owner, the group, and all others were allowed to read, write, and execute the directory. Such a configuration is very dangerous, since anyone can add a new user or group, or delete a user or group.

*Example 12.* In earlier NetWare systems, it was not difficult to forge NCP packets transmitted between clients and server(s). Therefore, Novell introduced the concepts of packet signatures in version 3.12. Unfortunately, this feature was not activated by default. In a default installation of the most recent version of NetWare, i.e., version 5, packet signatures will be used if requested by the other side. This implies that, if both the clients and the server(s) use a default installation, packets will not be signed.

## 5 Summary of Papers

The five appended papers are grouped into two parts. Part I consists of two papers. The first emphasizes security basics, i.e., concepts, definitions and terminology. It attempts to create a conceptual basis for the generic modeling of vulnerabilities that is the major objective of the thesis. The second paper presents the experimental campaigns that were carried out in order to gather intrusion data that we have used for analysis and modeling. The three papers of Part II focus on security analysis of general-purpose operating systems. [Paper C] and [Paper E] analyze the Windows NT and Novell Netware operating systems, respectively, while [Paper D] compares the features of Windows NT and UNIX. A summary of each appended paper is given below.

### 5.1 Part I: Security Basics – Definitions and Input Data

**A. On the Functional Relation Between Security and Dependability Impairments**

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 for 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.

B. IT Security Research and Education in Synergy
This paper presents experience from laboratory projects performed by students in Applied Computer Security, a course given at Chalmers University of Technology. From this, we conclude that the combination of security research and education results in a synergy that can be very beneficial to both fields. The paper describes in detail three different types of laboratory projects: intrusion experiments, intrusion analysis and remediation, and intrusion detection. Furthermore, we discuss the outcome and impact of the student projects with respect to education, research, and synergy between the two, as well as ethical issues. Among the benefits of the close connection between research and education in the projects, we found that the students were very motivated by this research connection and that they produced unique research data with a natural diversity.

5.2 Part II: Security Analysis

C. An Analysis of the Security of Windows NT
This paper presents an analysis of the security of Windows NT 4.0. The primary objective of the analysis was to investigate how secure the operating system actually is. A technical overview of the system, and in particular of its security features, is given. The system security was analyzed and practical intrusion attempts were made in order to verify known vulnerabilities and find new ones. The paper concludes that there are ample opportunities to improve the security of Windows NT.

D. A Comparison of the Security of Windows NT and UNIX
This paper presents a brief comparison of two operating systems, Windows NT and UNIX. The comparison covers two different aspects. First we compare the main security features of the two operating systems and then make a comparison of a selection of vulnerabilities, most of which we know have been used for making real intrusions. We found that Windows NT has slightly more rigorous security features than “standard” UNIX, although the two systems display similar vulnerabilities. The differences that we might indeed experience are believed to be due to the fact that Windows NT has not been on the
market as long as UNIX. Thus, the conclusion is that there is no significant difference between the “real” levels of security of these systems.

E. Security Evolution of a Network Operating System
In this paper, we study the security evolution of Novell NetWare from version 3.12 until the most recent current version, i.e., version 5. A detailed description of both security features and flaws in Netware 3.12 is presented. The flaws were revealed during an intrusion experiment performed by undergraduate students. The paper also deals with new security features, as well as newly discovered weaknesses, in versions 4 and 5 of the network operating system. The paper concludes that the weakest points in the Novell NetWare system are the clients themselves and the connections between clients and servers. Surprisingly enough, this remains true despite significant security improvements in later versions of the system. It is discussed whether this result can be generalized, leading to the conclusion that virtually no commercial system will ever be secure.

6 Discussion
This section discusses the underlying reasons for the existence of the vulnerabilities addressed in the thesis, and we make some general observations on their probable origin. All studied operating systems contain vulnerabilities. They can be discussed from various perspectives, from a very low technical level, through a system-related level, up to a very high organizational and sometimes even international level. Thus, a vulnerability usually exists at a number of different levels. This, in turn, implies that a vulnerability is not a single “atomic” phenomenon, but is rather composed of a number of sub-problems on the same or different levels.

When a flaw is discussed in the thesis, it is often presented on one level only, normally, a technical or system-related level. However, there is always some overall reason for it to exist, as discussed in the following.

There is no doubt that one of the major reasons for security flaws, and other software flaws, is to be found in the design phase. It seems that the “software crisis” from the 1960s is far from being over. Short-term profit interests, “time-to-market” demands, and insufficient organizational stability all weigh more heavily than security and software quality requirements. This, in combination with an ever-increasing software complexity, makes it virtually impossible to maintain an acceptable security level. Another type of complexity arises from the fact that, as noted in [Paper D], the trend is to re-use software, obviously in order to reduce development costs. This is done at the expense of security, since the adaptation of a software module into a new environment is by no means a trivial task. On the
contrary, it often gives rise to new and, in many cases, quite subtle and hard-to-detect security flaws. As Schneier stated in [38]:

“Security is a lot more than plugging in an algorithm and expecting the system to work.”

A further reason for insecure operating systems that must not be disregarded, is sheer incompetence. System designers do not always have sufficient knowledge in the programming language they are using nor in the development environment or the tools they use. Often, there is also a lack of understanding of security requirements and the reasons why these requirements exist. Here, we realize that we have an enormous educational task in front of us. Security education must be substantially improved and expanded at all levels from, at least, the undergraduate to the graduate levels. It is not until we reach a general public awareness of the security problems that system owners will refuse to deal with systems with insufficient security. This problem is in part addressed in [Paper B].

Furthermore, many security vulnerabilities are caused by the vendor’s ambition to be backward compatible with earlier versions of their system, even though these (older versions) have been proven vulnerable. This leads to a preservation of the insecurity of previous operating system versions and prevents major security improvements. Such problems are discussed in all the papers in Part II. The vulnerabilities described in examples 4, 8, and 12 in Section 4 are all caused by backward compatibility requirements.

A further observation is that we deal only with “steady-state security”, i.e., our requirements, definitions and understanding of the concept only regards systems under normal operating circumstances. It is difficult to address security during exceptional or dynamic conditions, such as (re-)configuration, bootstrapping, downloading of components, overflow, etc. and during various kinds of exception handling procedures. Thus, even if we designed our systems with a reasonable “steady-state security”, we would be far from achieving what we could call “dynamic security”. Here, a great deal of further research is needed, especially in light of the fact that the development is now so rapid that no steady-state exists.

Finally, it should be noted that available operating systems, with few or no exceptions, are Commercial Off-The-Shelf products, so-called COTS. COTS products suffer a number of deficiencies as pointed out in [27], the most important being the problem of verification, i.e., ensuring the functionality of the software that is being acquired, and in many cases downloading over an insecure network. In other words, you have no chance of knowing what you get when you purchase a COTS product, especially not if you download it from the public domain.
7 Results

The work presented in this thesis represents a further step towards a full understanding of the generic weaknesses and vulnerabilities that impair commercially available operating systems. Such an understanding is essential for the work with making systems secure, or at least sufficiently secure. To this end, we have suggested a terminology and a novel conceptual model that not only include security aspects but takes a broader perspective and integrates all types of dependability concepts into a new paradigm. So far, it has not been possible to fully apply the new paradigm, partly because further research remains to be done, e.g., addressing distributed systems as well as modeling system interaction and system hierarchies. Despite this fact, we have improved our knowledge and understanding of the weaknesses in the studied operating systems: UNIX, Windows NT and Novell Netware. We have also succeeded in structuring these vulnerabilities with the help of available taxonomies and finding common causes from two or more systems. It is expected that this work, when fully accomplished, will comprise a powerful basis for improving the security of operating systems, at least to the extent that research results are taken into consideration by the software developers. Unfortunately, this is not always the case as discussed in Section 6. In many cases, commercial and other considerations are given priority at the expense of security and there is a clear risk that sooner or later the final cost for this insecurity will have to be paid by the customer.

Acknowledgments

I would like to thank my supervisor, Erland Jonsson, for giving me the opportunity to join the computer security group at Chalmers and for being such an outstanding mentor. Thanks also to the other co-authors of the publications included in this thesis and the other members of the security research group (in alphabetical order): Dan Andersson, Stefan Axelsson, Ulf Gustafson, Hans Hedbom, Håkan Kvarnström, Ulf Lindqvist, Emilie Lundin, Lars Strömberg, and Helén Svensson. It has been a great pleasure to work with you all.

I would also like to thank my wife Helena, my brother Peter and Janet Vesterlund for proofreading various versions of the manuscript. Their comments and suggestions have been invaluable.

Finally, I would like to thank my wife Helena and my children Caroline, Sofia, David and Johanna for their never-ending love and support. To them I dedicate this thesis.
References


PART I

Security Basics – Definitions and Input Data
Paper A

On the Functional Relation Between Security and Dependability Impairments

Reprinted from

Proceedings of the
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

{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 for 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, modeling, terminology.

1 The author is also with the Department of Computer Science, Karlstad University, SE-651 88 Karlstad, Sweden.
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 [15] 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 re-definition 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 [21]. 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, 13, 14, 19, 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 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 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 [11, 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 [15]. 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 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].

SECURITY/DEPENDABILITY

PROTECTIVE ATTRIBUTE
environmental influence:

BEHAVIOURAL ATTRIBUTES
system behaviour:

THREAT
fault introduction
integrity

OBJECT SYSTEM
delivery-of-service
reliability/availability (safety)
denial-of-service
confidentiality/exclusivity (safety)

INTEGRITY

CORRECTNESS

TRUSTABILITY

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

\[1\] In the paper referred to, we used the term “preventive”, but “protective” seems to be a more adequate word.
etc. Although the word trustability 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.

**Figure 2.** The stages of security/dependability impairments

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 [16]. A third alternative for the definition of fault is found in [26] 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, 10].

A similar approach to that of the software community is found in a work on classification of software vulnerabilities [20]. 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 [20]. 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 sys-
tem, the state is *internal*; otherwise, i.e., if it is also a function of the behaviour with respect to the environment, 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—one 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 [18], 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.

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.

**Figure 3. Summary of terminology**

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.

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.

Table 1. Types of impairments

<table>
<thead>
<tr>
<th>Stage</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS-ENV interaction</td>
<td>Environmental influence</td>
<td>System protection</td>
<td>System operation</td>
<td>System behaviour</td>
<td>Impact on environment</td>
</tr>
<tr>
<td>Impairment propagation</td>
<td>Fault generation</td>
<td>Fault introduction</td>
<td>Error propagation</td>
<td>Failure occurrence</td>
<td>Failure consequence</td>
</tr>
</tbody>
</table>

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 the victim 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 [23]. 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 received 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 [23].

### 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 writable. 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 to 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 [22]. Similarly, real failures are better described as gradual degradations than as state changes [17]. 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.
References


PAPER B

IT Security Research and Education in Synergy

Reprinted from

Proceedings of the 1st World Conference on Information Security Education, WISE 1
IT Security Research and Education in Synergy

Stefan Lindskog¹, Ulf Lindqvist, and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden

{stefanl, ulfl, erland.jonsson}@ce.chalmers.se

Abstract

This paper presents experience from laboratory projects performed by students in Applied Computer Security, a course given at Chalmers University of Technology. From this, we conclude that the combination of security research and education results in a synergy that can be very beneficial to both fields. The paper describes in detail three different types of laboratory projects: intrusion experiments, intrusion analysis and remediation, and intrusion detection. Furthermore, we discuss the outcome and impact of the student projects with respect to education, research, and synergy between the two, as well as ethical issues. Among the benefits of the close connection between research and education in the projects, we found that the students were very motivated by this research connection and that they produced unique research data with a natural diversity.

Keywords: security education, computer security, intrusion experiments, intrusion analysis, remediation, intrusion detection.

¹ The author is also with the Department of Computer Science, Karlstad University, SE-651 88 Karlstad, Sweden.
1 Introduction

Most networked computer systems are continuously exposed to intrusion attempts. Even worse, some of the attacks succeed because people do not know how to implement system security. Unfortunately, the number of skilled practitioners of computer security today is small [10], making it necessary to incorporate security studies into educational programs. Furthermore, students must be prepared for real situations. Thus, practical laboratory experience is essential [11].

Since 1993, a course in Applied Computer Security has been given at Chalmers University of Technology, offered in the final year of the Master’s Degree Program. Topics covered in the course are: access control, authentication, cryptography basics, database security, network security, operating system security, physical security, privacy and ethical issues, program security, risk analysis, security management, security models, and security policies. An important component of the course is a laboratory project. During the first years of the course, the students were asked to evaluate the security of a target system by means of attacking it. The results of those experiments have been presented in many research papers, including [14].

The synergy between our research and education in security has been very fruitful for both fields. It is obvious that our research would not have been what it is today without the Applied Computer Security course. This elective has been much appreciated, and students have claimed that it has given them a “hands-on” feeling for computer security and contact with the research frontier.

In [12], Jonsson and Janczewski presented a survey of practical security experiments, conducted at a number of universities on most continents, including Europe, North America, Africa, Asia, and Australia/Oceania. The aim of the study was to investigate the role of experiments in information security education. They also suggested a taxonomy for such experiments. They found that many of the exercises and projects reported were different in terms of degree of applicability, degree of innovation and generalization. Moreover, the duration and effort (in man-hours) of performing an activity ranges from a duration of two hours with an effort of two hours to a duration of three months with an effort of 100 hours.

In the following, Section 2 gives some background information. The feasibility study is further described in Section 3, the first full-scale experiment is presented in Section 4 and other intrusion experiments conducted at Chalmers are briefly discussed in Section 5. Section 6 introduces a new approach called intrusion analysis and remediation, and Section 7 focuses on the intrusion detection assignment conducted in 1998. Section 8 discusses ethical issues, synergy effects, and educational and research aspects. Section 9 concludes the paper.
2 Background

In 1992, an EU research project was started with the intent of finding quantitative measures of operational security. For that purpose, empirical data was needed. To gather such data, practical intrusion experiments were conducted.

For the first experiment [2], which is referred to in this paper as the feasibility study, the primary goal was to investigate whether it would be possible to gather data for security modeling. This study was successful in many respects. First, we found that students are indeed able to break into a standard system within a limited time period and that they can be used in this type of experiment. Second, we learned how to set up such experiments and how to act as coordinators.

The first full-scale experiment [19] was done half a year after the feasibility study. In this case, our goal was to gather enough data for a quantitative security-modeling attempt. Final year students were engaged as attackers. All were at the same time taking the Applied Computer Security course, and the experiment was performed as a project in the course.

In both the feasibility study and the first full-scale experiment, the target system was a networked UNIX system. Three full-scale experiments have followed. On one occasion, a PC network was used as the target system [5, 6].

In the last couple of years, new ideas have been developed primarily to the benefit of education and secondarily to research. In 1997, for example, we introduced the remedy dimension in the project work. That time, the students were asked to suggest how a given vulnerability, which students found in the earlier experiments, could be eliminated from the system. In 1998, our increased research interest in intrusion detection was reflected in the course. The students would construct a small intrusion detection system using an expert system tool.

3 Feasibility Study

The feasibility study [2] (sometimes referred to as the pilot experiment) was intended as a pure research activity and conducted during the spring of 1993. The goal was to determine whether it would be possible for students to break into a normal system and how data from these attacks could be collected for quantitative security modeling.

3.1 Experimental Set-up

The target system was a networked UNIX system running SunOS 4.1.2, consisting of 22 disk-less workstations and a file server, all configured as recommended by the vendor.
Standard accounting was enabled on all computers for monitoring purposes and, when the experiment was performed, the system was in regular use by other students. Finally, the security of the system was that of a default installation and was not improved in any way.

Three different types of actors were identified: the attackers, an experimental coordinator and the normal system administrator. Members of the Chalmers University Computer Club adopted the role of attackers, as it was expected that these students have greater knowledge of security vulnerabilities than other students. Only 13 students participated in the first experiment, and each attacker was a legal user of the system, i.e., an insider. One researcher acted as experimental coordinator, monitoring and coordinating all activities in the experiment. Finally, the normal system administrator represented the system owner. He was instructed to behave as normally as possible.

The attackers were required to follow a few rules, which were presented in a briefing before the experiment began. The main reason for these was to encourage the attackers to behave as realistically as possible. For example, they were not allowed to cooperate with other attackers during the experiment or to cause physical damage to the hardware.

Before the experiment took place, each attacker was required to fill in a questionnaire that described his/her formal education, prior experience in the area, etc. In addition, for each intrusion attempt, an activity report was written. Thus, working time, on-line time, resources used and the occasion at which the activity was performed were documented. At the completion of the experiment, every attacker filled in an evaluation report. The activity and evaluation reports were both standardized, fill-in forms. Finally, the attackers were asked to write a final report to give their views of the experiment, summarize their activities, describe successful and unsuccessful attacks and ideas for other attacks, etc., and to describe the outcome of the experiment from a personal point of view.

3.2 Results

With respect to data collection methodology, the most important result was that students could indeed be used in experiments such as this. However, the amount of data received was, as expected, too sparse to be used for any statistical modeling.

This experiment also taught us a great deal about the intrusion process. Most notable is that, by measuring the effort needed to break into a system, we might in the future be able to quantitatively determine the security level of a particular type of computer system. Another interesting conclusion is that it is even more difficult to measure the reward, i.e., the motivating force, of a particular successful intrusion than measuring the effort. Finally, data collected in this study have been used in several research papers, including [2].

Problems that occurred were that attackers tended to leave the experiment for a lack of ideas and/or motivation and that the required reporting turned out to be too extensive. Still, this experiment was highly appreciated by the participants.
4 Full-Scale Intrusion Experiment

While the feasibility study was a pure research activity, the first full-scale intrusion experiment was a combination of research and education. The research goals were to collect enough data to make a quantitative modeling attempt possible and to learn more about the intrusion process. The educational goal, however, was to offer a realistic project in the Applied Computer Security course, which was given for the first time. The experiment was conducted in November and December, 1993.

4.1 Experimental Set-up

We used the same target system as in the feasibility study, with one security improvement. The workstations were set up to require a password to successfully perform a so-called single user boot-up sequence.

In this experiment, we engaged as attackers Master of Science students in Computer Science and Engineering in their final year. They worked in groups of two students, and a total of 12 groups attacked the system at the same time. We expected each group to spend approximately 40 hours of effective working time. The attackers were required to contact the coordinator once a week to discuss their progress. The same rules as described in Section 3 were to be followed.

In the feasibility study, the participants had complained about the excessive reporting, and several simplifications were thus made here. At the end, 65 security breaches of various kinds were reported, to be compared with the 25 breaches in the feasibility study.

4.2 Educational Results

The project was well received by the students. We are convinced that the results were very surprising to many of the participants when they discovered that it was not as difficult as they thought to break into a standard UNIX system. Indeed, most students claimed that the experiment gave them a “hands-on” feeling for computer security and raised their awareness of the problems involved.

4.3 Research Outcome

This study taught us even more about the intrusion process. Publicly available tools for searching and utilizing known vulnerabilities in the specific system, in this case SunOS 4.1.2, were used in several intrusion attempts. The groups downloaded these tools from the Internet. Further, more events were reported and the recording of data was more complete in this second experiment. The data collected has been used for research presented in several papers, including [13].
5 Further Experiments

Three additional full-scale intrusion experiments have been conducted. Thus, a total of five intrusion experiments in which students have adopted the role as attackers has been carried out at the Department of Computer Engineering. We summarize these in Table 1.

<table>
<thead>
<tr>
<th>Year</th>
<th>Experiment</th>
<th>Operating System</th>
<th># of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring 1993</td>
<td>Feasibility study</td>
<td>SunOS 4.1.2</td>
<td>13</td>
</tr>
<tr>
<td>Autumn 1993</td>
<td>1st full-scale intrusion experiment</td>
<td>SunOS 4.1.2</td>
<td>24</td>
</tr>
<tr>
<td>Autumn 1994</td>
<td>2nd full-scale intrusion experiment</td>
<td>Novell Netware 3.12</td>
<td>14</td>
</tr>
<tr>
<td>Autumn 1995</td>
<td>3rd full-scale intrusion experiment</td>
<td>SunOS 4.1.3</td>
<td>32</td>
</tr>
<tr>
<td>Autumn 1996</td>
<td>4th full-scale intrusion experiment</td>
<td>SunOS 4.1.3</td>
<td>42</td>
</tr>
<tr>
<td>Autumn 1997</td>
<td>Intrusion analysis and remediation†</td>
<td>SunOS 4.1.3</td>
<td>59</td>
</tr>
<tr>
<td>Autumn 1998</td>
<td>Intrusion detection‡</td>
<td>SunOS 4.1.3</td>
<td>87</td>
</tr>
</tbody>
</table>

† Intrusion analysis and remediation are described in Section 6.
‡ Intrusion detection is covered in Section 7.

In 1994, the target system was a PC network system. Insecure clients were running DOS 6.2 and Windows 3.1, and the network file server was configured with Novell Netware 3.12. Despite the fact that only one student was familiar with Novell Netware, the target system was easily and successfully attacked and breached by all students except one. In most cases, they exploited the insecurity of the clients in the sense that any user had full access to everything at the client. The experiment is further described in [6], and [5] presents an interpretation of the intrusion data collected.

In the third and fourth full-scale experiments, a networked UNIX system, this time with SunOS 4.1.3, was again used as the target system. In Table 1, we can see that the number of attackers is increasing, and our experience is that our approach to intrusion experiments scales well to the number of attackers in terms of the educational resources required. However, when the number of attackers increases, the compilation of recorded data for research purposes becomes much more difficult because most of the work needs to be done manually.

Two other experiments should also be mentioned. First, procedures inspired by the intrusion experiments were used in an analysis of a secure system based on trusted components. See [16, 17] for a detailed description of this experiment. Second, a similar procedure was used to evaluate the security of Windows NT; see [7, 8] for an in-depth description. These two experiments differ from those described earlier in one important aspect: we did not engage students as attackers. Instead, we adopted the role as attackers, presumably more of tiger team attackers [1, 4, 9], ourselves. In the latter study, we found that Windows NT can be penetrated quite easily. This implies that Windows NT can be
used as a candidate target-system in future intrusion experiments in which students act as attackers. One such experiment will be conducted at Karlstad University in the spring of 2000.

6 Intrusion Analysis and Remediation

In the previous experiments, students were asked to find as many weaknesses as possible in the target system by attacking it. In 1997, after four full-scale experiments, we were interested in trying something new. The new approach was to let the students act as though they were on the other side, i.e., to act as the system owner. This experiment was primarily an educational activity in the sense that we did not foresee any specific research outcome.

6.1 Experimental Set-up

The target system was the same networked UNIX system as was used in the third and fourth full-scale intrusion experiments described in Section 5. A list of 13 carefully selected intrusions found in them was presented to the students. The class of 59 students was divided into groups of two students each, i.e., 30 project groups were formed. This implies that several groups analyzed the same intrusion. Each project group selected one intrusion and was asked to:

1. Reproduce the selected intrusion, verifying that it can be used to circumvent the security of the target system
2. Describe the weakness and explain why the intrusion works
3. Suggest remedies that would make the system secure against this kind of intrusion

The only intrusion descriptions they were given were those written by the previous students in intrusion experiments. Furthermore, each intrusion was assigned an estimated level of difficulty. Roughly, three levels were used—simple, medium and complicated. The groups were required to document their work. In fact, two reports were compulsory: a complete project report and a summary report of approximately two pages. They had to describe the vulnerability according to the taxonomy suggested by Landwehr et al. [15].

Thus, they needed to answer the following questions:

– How did the vulnerability enter the system?
– When did the vulnerability enter the system?
– Where in the system does the vulnerability manifest?
When the exercise was finished, copies of the summary reports were distributed to all participants to give the students a broader view of intrusions than the specific case they had studied. In addition, each group presented its work to the class.

6.2 Results
The students were put in a situation like one many of them will probably experience in their future careers as either software engineers or system administrators. Most students reported that they found the exercise interesting and valuable. It is also of interest to note that some groups had great difficulty in reproducing the intrusion. In some cases, we had been overly optimistic as to the programming skills and computer literacy of the students and, in other cases, the reason was probably that it was impossible to perform the intrusion on our system. The exercise required far more supervision than we had expected.

7 Intrusion Detection
In 1998, the steadily increasing number of students in our course had forced us to give all groups identical assignments, as in a traditional laboratory exercise. To retain the connection with our research, we decided to let the students use a software tool that had previously been used only by experts to construct research prototype intrusion-detection systems. The tool, called P-BEST, is further described in [18].

7.1 Experimental Set-up
The assignment was to construct a system that could be used to automatically detect attacks against a file transfer (FTP) server. The students were given a tool designed for building expert system components of intrusion detection systems. The tool is the same one we use in our research, and the exercise would support or contradict our hypothesis that the tool is easy to use for beginners. It was required of the students that they include in their lab reports a discussion of their experiences with using the tool.

For evaluation of the system they had designed, the students were given a very large data file containing recorded network data representing actual FTP transactions. A few real and synthetic intrusions were mixed with a large number of normal transactions, and the students used their system to find those intrusions.

7.2 Results
A total of 87 students participated in the assignment, with a few exceptions working in pairs, making a total of 46 groups. Of those 46 groups, 25 had constructed a system that
gave a completely correct answer. An additional number of eight groups would most likely have obtained the correct result if they had not misinterpreted a vaguely formulated part of the instructions. Only a handful of groups failed to hand in a report before the given deadline. Most students reported that they found the exercise interesting, and some even took the time to give detailed suggestions for improvements to the tool.

8 Discussion

The outcome and impact of our student projects can be viewed from several different perspectives. This section discusses aspects of education, research and synergy between the two, as well as ethical issues.

8.1 Educational Aspects

Probably the most important result from an educational perspective is that the laboratory exercises, described in Section 4 through 7, give a practical view of security. The students have learned among other things how an intruder thinks and become aware of common threats to today’s networked computer systems. We therefore believe that the assignments have prepared them for forthcoming real situations. In addition, comments by the students about the exercises have been very positive. Two key words often found in the evaluation reports are “fun” and “exciting”. Several students claimed this to be the best course they have taken. Furthermore, students usually have very little contact with research during their studies, and this has made our course different and interesting.

8.2 Research Aspects

There are several concerns as regards the validity and accuracy of the data obtained from student exercises. The first question is whether the students in our experiments constitute a good approximation of real attackers. In reasoning about security, the discussion often quickly reaches the point where the goal is to protect the system against the most skilled and powerful attacker in the world. The term “übercracker” [3] has been used for this picture of a diabolic, omnipotent adversary. We did not wish to investigate the übercracker, partly because most people in the security community are already doing that. We wanted to see how ordinary computer users with academic training in computer science and engineering—but no previous attacker experience—would operate when they suddenly had a reason to attack the system on which they were working. We found that unskilled attackers can indeed perform technically advanced attacks, thanks to so-called exploit scripts which they download from the Internet [16]. An exploit script is a program designed and pub-
lished by a skilled attacker. When executed, it will automatically carry out an attack on the target system. We claim that we have successfully modeled the infamous insider threat and that our results are of general value [14].

The second question is how well the students’ reports correspond to reality. The set of actions actually carried out by the attackers in our experiments is largely in agreement with the set of actions documented by the attackers in their reports, but the sets are not identical. There are indeed actions reported that we seriously doubt were ever performed on the system, for example types of intrusions to which we know that the system was not vulnerable. This became painfully clear to the students who later tried to confirm the reports in the intrusion analysis exercise. It is also likely that there are actions that the attackers performed but never reported to us. Why is there not a perfect match between the two sets? For the first case, we do not really know what reason the students have for exaggerating their results. The grading of the reports was not based on their successes but on their efforts and their analysis of their work. We had also made it clear that their reports would be used as research material, and it is therefore disappointing and worrisome that some of them tried to polish their results. For the second case, we can only hope that actions left out of the reports were not reported because the students considered them insignificant and not because they wanted to hide something from us.

Finally, there is the problem of using a relatively large group of humans as “guinea pigs” at an engineering department. Our expertise is in technology, not in the behavioral sciences. Factors not related to technology can greatly affect the results, for example, how instructions are interpreted, what motivation the participants have, whether they obey the rules of the experiment, whether they tell the truth, etc. This is probably an area in which we could benefit from consulting external expertise.

8.3 Synergy Effects

To understand the benefits of conducting this kind of research and education together, we can reason hypothetically about having one without the other. First, we consider having education without research. Before our research group was formed, security was only briefly introduced to students as a part of other courses, for example, operating systems, computer communication, and computer networks. The growing general interest in security had probably led to the introduction of a separate course in security sooner or later. However, security is a demanding topic for teachers because it is a rapidly moving field in which it is difficult to remain up-to-date—not only because of the sheer amount of information but also because of obscurity, misunderstandings, and a lack of perspective in many bulletins. Teachers must also have a thorough knowledge of the fundamental principles and lessons often forgotten today. Without the genuine interest and devotion to the topic that active members of the research community have, it is much more difficult to give high quality education in security.
Second, we can consider research without education. We have already argued how valuable the data produced by the students has been for our research. Computer science students are often considered a homogeneous group, but our student experiments have actually helped us to create a diversity in our data that we would not have been able to achieve on our own. Students are also naturally willing to learn and therefore ask questions that less motivated participants would not bother with.

8.4 Ethical Issues
In the remainder of this section, we take an external view and discuss our research methods and their consequences as they are seen outside the security research community.

8.4.1 The Hacker School
One concern that was voiced by several sources when our experiments came to public knowledge was that we were training university students to become computer criminals. This was amplified by some reports in newspapers and television. Fortunately, the spectacular idea of a television talk show host to broadcast attacking activity live from our computer lab was dropped a couple of days before it was to go on the air for reasons not having to do with our activities. A message that did reach the public, and still haunts us several years later, was that Chalmers had attacked hospital computers. The journalistic logic behind this conclusion was that, because we had performed our first experiment on a UNIX system and the hospital administration in a nearby county had this type of system, hospitals could probably be attacked in a similar way and, ergo, Chalmers had trained students to attack hospitals.

What are the facts in this case? In the intrusion experiments, we refrained entirely from training students in attacking the system, which was necessary to ensure that the data collected was valid for security modeling. Some students complained about the lack of hints and instructions from us, since we simply gave them access to a computer network which they were allowed to attack for a limited period of time. We also gave them access to the Internet, where they could seek information. In 1993, this was not something that students could normally arrange on their own. Today, any student can afford to set up a computer network in his or her home, consisting for example of a number of “old” PCs with an Intel 486 processor running Linux. Internet access is something that most people take for granted today. This change is reflected in our approach to the remedy analysis, where we in fact did tell the students how they could perform attacks. In 1997, this was a much less controversial issue because all the attacks had been discussed publicly on the Internet and because anyone could try it at home, as explained above. All along, we have informed the students about what constitutes computer crime according to national laws, and why certain behavior is inappropriate or illegal.
8.4.2 Informed Consent
An important concept in research ethics is informed consent, which means that experiment participants should be informed of all possible consequences and risks of the experiment, should understand that information, and should be given the possibility to refuse participation. In the intrusion experiments, the students who played the role of attackers were of course informed about the experiment but were not informed that they could refuse and be given another assignment instead, simply because we assumed that they all wanted to participate.

The situation was different for the students who were ordinary users of the system. They were not informed about the experiments because we did not want them to be more concerned about security than they normally are. Unknowingly, they played the role of victims. Some were deceived by forged e-mail to send the attackers their passwords, others had their passwords revealed in other ways and, worst of all, some passwords to other systems were monitored by the attackers when lab computers were used as terminals for remote access. This is perhaps the most questionable part of the experiment, with a trade-off between realism and ethical concerns. The users were not as easily fooled in the later experiments because they had heard about our previous activities. In fact, any computer malfunction was blamed on the security experimenters.

In the intrusion detection exercise, we wanted to use primarily real recorded data to make the students feel the realism and relevance of their assignment. Users accessing the monitored FTP server had been warned through a login banner message that their activities were monitored, and the passwords for non-anonymous users were never recorded. Still, there can be innocent users whose transactions appear suspicious in the log file or other privacy concerns. Again, there is a trade-off between realism and ethics.

8.4.3 Related Discussions on Ethics
In a discussion of the ethical aspects of spreading information on methods for computer crime, computer crime expert Donn Parker claims that the intent of the publisher is what matters [20]. The director of security research at Purdue University, Eugene Spafford, shares this view [21]. If the intent is to raise awareness and protect systems, then it is ethical (and legal). If the intent is to encourage people to attack systems, then it is unethical and probably illegal.

At the 21st National Information Systems Security Conference in Arlington, Virginia, in October of 1998, there was a panel discussion entitled “Do attack/defend exercises belong in the classroom?” It is interesting to note that all panelists from academia were in favor of such exercises as part of security education and that it was difficult for the organizers to find a panelist with the opposite opinion. One concern brought up in the discussion was that students could use skills acquired in the exercise for evil purposes. This is however a general concern that is not restricted to computer security; any tool or skill can be
used to do evil. A similar view is presented by White and Nordstrom [22], who claim that it would be more dangerous not to educate future system administrators in the details of attacking techniques, because they would otherwise be “sitting ducks” for attackers who possess these skills.

9 Conclusions

We have presented how we have successfully used students to produce security research data while at the same time educating the students in applied computer security. The close connection between research and education has been a motivating factor for the students, and our view of the students as a valuable research resource has increased our motivation in our role as teachers. At the same time, we have collected valuable research data that could not easily have been acquired in other ways. In many cases, the dual role as teacher and researcher at a university leads to conflicts of interest, but we have managed to turn that duality into a fruitful synergy.

Acknowledgments

We are grateful to all the students that have participated in the experiments throughout the years.
References


PART II

Security Analysis
PAPER C

An Analysis of the Security of Windows NT

Reprinted from

Technical Report 99-16
Department of Computer Engineering
Chalmers University of Technology,

An earlier version appeared in
Proceedings of the Second Nordic Workshop on
Secure Computer Systems, NordSec ’97,
Espoo, Finland, November 6–7 1997.
An Analysis of the Security of Windows NT

Stefan Lindskog\textsuperscript{1}, Hans Hedbom\textsuperscript{1}, and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
{stefanl, hansh, erland.jonsson}@ce.chalmers.se

Abstract

This paper presents an analysis of the security of Windows NT 4.0. The primary objective of the analysis was to investigate how secure the operating system actually is. A technical overview of the system, and in particular of its security features, is given. The system security was analyzed and practical intrusion attempts were made in order to verify known vulnerabilities and find new ones. The paper concludes that there are ample opportunities to improve the security of Windows NT.

Keywords: security analysis, vulnerability, intrusion, Windows NT.

\textsuperscript{1} The authors are also with the Department of Computer Science, Karlstad University, SE-651 88 Karlstad, Sweden.
1 Introduction

Computer security is traditionally defined by three attributes, confidentiality (or secrecy), integrity, and availability [14, 20], also known as the “CIA”. Confidentiality is the aspect of keeping information protected from unauthorized users. Integrity implies that data can be modified only by authorized parties. Finally, availability means that information is provided to the authorized user.

The security of a particular system has traditionally been evaluated according to some kind of criteria [14, 25]. Such methods typically have a number of predefined security classes with different requirements. TCSEC, for example, has seven classes, D, C1, C2, B1, B2, B3 and A1, where class A1 has the strongest security requirements. A system that is evaluated for, say, class B2 but does not fulfil one (or more) requirement(s) will automatically be put in class D. Another common evaluation method is called tiger team penetration testing [3, 8, 12]. A group of people very skilled in the computer security area attacks a system. The idea is that these people will, with a high probability, find most of the system vulnerabilities.

In this paper, we analyse the security of Windows NT 4.0 with Service Pack 3 (SP3) installed, which is claimed to have been designed with security in mind. In fact, it has been classified as a C2

level operating system by the National Security Agency (NSA) [18]. The objectives of the present study were twofold: first, to learn more about the security subsystem in Windows NT, and, second, to identify as many vulnerabilities as possible within a limited time period.

There is not a great deal of in-depth literature on Windows NT and its security design. A number of security handbooks have recently been published [15, 21, 22, 24], but unfortunately, they all lack technical depth. The evaluation report from the National Computer Security Center (NCSC) [18] does reach an acceptable technical level, although the evaluation is based mostly on the design of the system as defined in the Orange Book [25]. Here, we studied the operational security of Windows NT. Practical intrusion attempts were made in order to reveal vulnerabilities in the system.

Other penetration experiments have been carried out at our department, the most similar being a security analysis of a secure database [16, 17]. Most of our previous studies differ, however, from the present. Firstly, the object systems were different: a networked UNIX operating system [5, 19] and a PC network [9, 10]. Secondly, the attackers were final year university students. Thirdly, the attackers had to follow some rules, since the results of these studies were meant to be used for mathematical modeling purposes.

---

1 Recently, Windows NT also passed an E3 evaluation according to ITSEC [14].
In the following, Section 2 gives a system overview of Windows NT, while Section 3 presents the security features. In Section 4, vulnerabilities found during penetration testing are described. Section 5 discusses impressions and trends and concludes the paper.

2 System Overview

Windows NT was developed by Microsoft Inc. and was first released in 1992. It was expected to replace Windows 3.x. Windows NT is, unlike Windows 3.x, a full-fledged 32-bit OS with support for processes, multiple threads, symmetric multiprocessing, and distributed computing. The system is designed to be able to execute on a variety of hardware platforms, e.g., Intel 386, Alpha AXP etc. It is implemented mainly in C with the exception of the software that communicates directly with the hardware, which is written in assembler. Windows NT uses an object model to manage its resources, and the term “object” is thus used instead of “resource”. Microsoft claims that Windows NT is designed to be a secure operating system—one of their goals was to meet the C2 evaluation criteria. Such systems must implement discretionary access control at the user level and must provide mechanisms for tracking all accesses (or attempted such) to an individual object [25]. Another C2 level requirement is elimination of residue exposure. Furthermore, Windows NT has a new user interface (Win32). It is designed to give the user the impression of working in the familiar Windows 3.x environment but, behind the façade, Windows 3.x and Windows NT are totally different.

2.1 Architecture

The Windows NT architecture is a hybrid between the layer model and the client/server model [7]. The former model is used in the executive, which is the part of the operating system that executes in kernel mode (supervisory mode), see Figure 1.

The client/server model is used in the top-most part of Windows NT. A server, sometimes called protected subsystem, provides either an execution environment for applications or is an integral part of the operating system, see again Figure 1. Win32 is an example of a server that provides an execution environment for Win32 applications. The security subsystem, which in reality is implemented as a set of servers, implements the security policies on the local computer. An example of a task that the security subsystem handles is user logons.
2.2 The Executive

The executive offers basic operating system functionality, or services, to protected servers. These services are implemented as a set of managers or monitors. Hence, a process manager, a memory manager, an I/O manager, an object manager, and a security reference monitor are all examples of services provided. In addition, there is also a kernel and a so-called *Hardware Abstraction Layer* (HAL) in the executive. These two components implement the lowest-level functionality of the executive.

2.3 Protected Subsystems

Protected subsystems are servers running in user mode with certain privileges. The *Session Manager*, the *WinLogon*, the *Win32*, the *Local Security Authority* (LSA), and the *Security Accounts Manager* (SAM) are five important and necessary standard servers in Windows NT. A short summary of these are given below.

*Session Manager* is the first server to start in a Windows NT system. It is responsible for loading DOS device drivers and subsystems registered in the Registry\(^1\), and for initialization of Dynamic Linked Libraries (DLLs), after which it starts the WinLogon server.

*WinLogon* is the logon process. It is responsible for coordinating and providing interfaces for interactive logons.

---

\(^1\) The Registry is a collection of system information for the actual system.
Win32 makes Microsoft’s 32-bit Windows API available to application programs. In addition, it provides the graphical user interface and controls all user input and output.

LSA (Local Security Authority) has its main responsibilities centered around security. It plays a major part in the logon procedure as well as in upholding the security policy for the local system.

SAM (Security Accounts Manager) is responsible for managing account information for users and groups, either locally or domain-wide depending on its role. Account information is stored in a database, which is accessible only by SAM. The database is referred to as the SAM database.

### 2.4 File Systems

Windows NT has built-in support for a number of file systems, including the File Allocation Table (FAT) file system, the High Performance File System (HPFS), and the new Windows NT file system (NTFS). The FAT file system was originally designed for the MS-DOS operating system, while HPFS was designed for OS/2 1.2, mostly to support the LAN Manager file server. NTFS, on the other hand, is a completely new file system for Windows NT. A detailed description of NTFS is given in [6].

### 2.5 Networking

To meet the expected market requirements for the 1990s, Microsoft added network capabilities to Windows NT. The key components in Windows NT’s network architecture are the redirector and the network server. The former is triggered when a user, or application, on a client machine requests a file on a remote machine. In such a case, the redirector will receive the request and forward it to the remote machine. The network server, on the other hand, resides on a remote computer, awaiting requests from network redirectors. The protocol used between these components is by default based on Server Message Block (SMB), which is a high-level specification for formatting messages that are to be sent over the network. From Windows NT version 3.5, the default transport protocol is TCP/IP.

#### 2.5.1 Domains

The management of user accounts in large networks is often a great burden, making centralizing management of user accounts desirable. Windows NT uses domains for that purpose. A domain consists of a primary domain controller (PDC) and zero or more backup domain controllers (BDC). All user management is carried out on the SAM database on the PDC and is replicated to the BDCs.
2.5.2 Trust Relationship
Trust relationship is another concept related to large networks. With trust, one domain server may authenticate a user to a domain server in another domain. It is used mainly to centralize user management in large companies. A trust relation is always one-way, i.e., one domain trusts another domain to authenticate users. In cases in which two-way trust is needed, two one-way trust relationships are established, one in each direction.

2.5.3 Shares
In a distributed environment, network resources are shared by many users. A shared resource in Windows NT must be exported. An exported resource is referred to as a share. A printer or a file system tree may be defined as a share.

3 Security Features

Windows NT provides a unified access control facility which applies to processes as well as other objects in the system. The access control is built around two components, an access token associated with every process and a security descriptor connected to every object where interprocess access is possible. When a user logs onto the system, he must provide a username and a password to authenticate himself to Windows NT. If he is accepted, a process and an access token are created for him. The access token is associated with the process. An important attribute in the access token is the Security ID (SID). The SID is an identifier that identifies the user to the system in areas concerning security. A process inherits the access token of its creator. There are two reasons for maintaining access tokens [23]:

- It keeps all the necessary security information in one place in order to speed up access validation.
- It allows every process to modify its own security attributes, in a restricted manner, without affecting other processes in the system.

Allowing the process to change its own security attributes may appear strange at first glance, but a newly created process is normally started with all attributes disabled. Later, when a thread wishes to perform an allowed privileged operation, the thread enables the necessary attributes before the operation is executed.

\[1\] “He” should be read as “he or she” throughout this paper.
3.1 Subjects and Objects
A subject is a process\(^1\) with one or more threads that execute on behalf of a user, and objects are all other \(\text{Windows NT}\) resources that can be accessed and used by a subject. The privileges, the user SID and the group SID of a user’s processes are stored in the access token held by the process. This token, called the primary token, is used to grant threads different types of access to different objects. Threads can also hold one additional token, called an impersonation token. This token, which allows the thread to act on that subject’s behalf, is given to the thread by another subject. The information in the token is compared with the information stored in the object’s Access Control List (ACL). The Security Reference Monitor (SRM) performs the comparison when the object is first opened by the process.

3.2 Users and Groups
Only legitimate users may access objects on a \(\text{Windows NT}\) system. Such a user must have a valid user account. In addition, in order to access a particular object, the user needs to have access rights to it. Guest and Administrator are the default accounts in all \(\text{Windows NT}\) systems.

Another supported concept is groups. With groups, permissions can be granted to a set of related users, which facilitates the procedure of granting rights and permissions. A user may belong to more than one group at the same time. There are a number of predefined groups in \(\text{Windows NT}\): Administrator, Backup Operators, Printer Operators, Power Users, Users, and Guest.

There are basically two types of accounts, global and local. A global account is accessible and visible inside a whole domain, while local accounts are accessible and visible only on the computer on which they were created. There are also two types of groups, local and global. Local groups can only be used inside the domain, while global groups can be used, and are visible, in all the trusted and trusting domains of the system.

3.3 Object Security
A security descriptor is associated with every named object in \(\text{Windows NT}\). It holds information about logging for and access to the object. The access control is implemented by an Access Control List (ACL). The ACL consists of a list of Access Control Entries (ACEs). An ACE contains a Security IDentifier (SID) and the operation that the owner of this SID is allowed (or disallowed) to perform on the object. ACEs that disallow operations are generally placed before ACEs that allow operations. The ACL is tied to the object when it is created in one of three different ways.

\(\text{Processes in Windows NT are indeed implemented as objects and are accessed using object services.}\)
When a process wants access to an object it has to open the object. In the open call, the process must specify what operations it intends to perform on the object. The system will then search the ACL of the object to try to find an ACE that contains the SID of the caller (or of any group of which he is a member) and allows or disallows the requested operations. The search is done on a “first match” basis. If the call succeeds, a handle to the object is passed to the process. All subsequent calls to the object will be made through this handle so no further checks are done as long as the process performs only the operations specified in the open call. If new operations are required, the object must be re-opened.

3.4 Logon Security
The logon procedure in Windows NT is fairly complicated and involves both the executive and a number of protected servers, including WinLogon, LSA, and SAM, to authenticate and grant privileges to a user. The WinLogon server is the coordinator. A detailed description of the logon procedure is given in [18].

At logon time, a user can choose to logon either locally or remotely. In the former case, the user is authenticated against the local SAM database, and in the latter case against a SAM database stored on a Domain Controller (DC). In remote logons, the netlogon service, which is started when the computer is booted, is used on both sides. Passwords sent over the network are encrypted.

3.5 Account Security
As mentioned in Section 3.2, only legitimate users are allowed to access objects on a Windows NT system, and each user belongs to one or more groups. User accounts, except Administrator and Guest, are defined by users that have the permissions required to create new accounts, e.g., users in the Administrator group.

3.5.1 The SAM database
All account information is centrally stored in the SAM database, which is accessible only by the SAM subsystem. Passwords are stored as hashed values and, by default, two hashed variants of each password are stored, in LAN Manager format and in NT native format.

3.5.2 User Rights
In addition to account information, e.g., username and password, a set of attributes called “rights” can be specified. They define what a user (or group) can do in a system. The set of rights is given in Table 1. Most of the rights are self-explanatory, for example a user account with the Log on locally rights specified is allowed to logon to the system from the local machine. Windows NT has a standard policy for assigning rights to certain groups.
For example, on a Windows NT workstation, everyone has the right to logon locally while, at a Windows NT server, only Administrators and Operators are able to do that.

### Table 1. User rights in Windows NT

| Access this computer from the network | Access as part of the operating system |
| Add workstation to domain            | Backup files and directories          |
| Bypass traverse checking             | Change the system time                |
| Create a pagefile                    | Create a token object                 |
| Create permanent shared objects      | Debug programs                        |
| Force shutdown from a remote system  | Generate security audits              |
| Increase quotas                      | Increase scheduling priority          |
| Load and unload device drivers       | Lock pages in memory                  |
| Log on as a batch job                | Log on as a service                   |
| Log on locally                       | Manage auditing and security log      |
| Modify firmware environment values   | Profile single process                |
| Profile system performance           | Replace a process level token         |
| Restore files and directories        | Shut down the system                  |
| Take ownership of files or other objects |                                      |

### 3.5.3 Account Policies

The account policies regulate password restrictions and account lockouts. These policies are first defined when the account is created, but may later be changed. The following password options can be set:

- **Maximal password age**, sets limitations for the length of time a password can be used before it must be changed.

- **Minimal password age**, sets limitation for the length of time a password must be used before a user is allowed to change it.

- **Minimal password length**, sets a limit on the minimum number of characters of which a password must at least consist.

- **Password Uniqueness**, where a Windows NT system can be configured to remember old passwords and force users to choose new passwords that have not recently been used.

An account can also be specified to be locked out after a given number of unsuccessful logon attempts. The lockout feature offers a few options:

- **Lockout after n bad logon attempts**, where n must be assigned a positive integer value.
– *Reset count after m minutes*, where m specifies how many minutes shall pass before the bad logon counter is cleared.

– *Lockout duration*, where forever (i.e., until an administrator unlocks) or duration in minutes are the possible choices.

Note, however, that the default administrator account cannot be locked in any way.

### 3.6 TCP Security

A secure system must permit only services that are proved secure and necessary [4]. In Windows NT, it is therefore possible to block communication to both TCP and UDP ports. This implies that a system can be configured to accept only packets sent to specific ports on which secure servers listen. Microsoft calls this feature TCP security.

### 3.7 Auditing

The Security Reference Monitor (SRM) and the Local Security Authority (LSA) are responsible for auditing in Windows NT together with the Event Logger. Different types of events are grouped into event categories, and auditing is then done on the basis of these groups. There are seven types of event groups [18]: System, Logon/Logoff, Object Access, Privilege Use, Detailed Tracking, Policy Change and Account Management.

The auditing is based on audit records constructed on request from the responsible subsystem by SRM (in some cases by LSA). Requests from the executive are always carried out, while servers need the Audit Privilege for SRM to honour their requests. The request must be sent for each occurrence of an event. The audit record is then sent to LSA, which in turn sends it to the Event Logger, after field expansions and/or field compressions. Finally, the Event Logger commits the audit record to persistent storage.

### 4 Security Flaws

This section presents our analysis of a networked Windows NT 4.0 system with Service Pack 3 (SP3). It was installed on standard PCs. Both server and workstation versions of the operating system were used. The analysis uncovered a number of vulnerabilities, most of which were also demonstrated in practice using so-called exploit scripts. An exploit script is a program designed and published by a skilled attacker. Such programs can very often be downloaded from the Internet and executed by an unskilled attacker.

The remainder of this section presents security flaws found in searching the World Wide Web and news groups on the Internet. Hence, these vulnerabilities are publicly
known. Below 13 different attack methods\(^1\), classified according to the CIA (confidentiality, integrity, and availability), are described and discussed in detail. Table 2 gives a summary of the attack methods.

<table>
<thead>
<tr>
<th>Category</th>
<th>Attack Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality attacks</td>
<td>C1 Extract encrypted passwords from the SAM database</td>
</tr>
<tr>
<td></td>
<td>C2 Mount and read an NTFS volume</td>
</tr>
<tr>
<td></td>
<td>C3 Remotely mount and read an NTFS volume</td>
</tr>
<tr>
<td></td>
<td>C4 Anonymous logon</td>
</tr>
<tr>
<td></td>
<td>C5 Crack passwords</td>
</tr>
<tr>
<td></td>
<td>C6 Capture passwords</td>
</tr>
<tr>
<td>Integrity attacks</td>
<td>I1 Mount and update an NTFS volume</td>
</tr>
<tr>
<td></td>
<td>I2 Remotely mount and update an NTFS volume</td>
</tr>
<tr>
<td></td>
<td>I3 Overwrite the administrator’s password</td>
</tr>
<tr>
<td></td>
<td>I4 Join the administrator group</td>
</tr>
<tr>
<td>Availability attacks</td>
<td>A1 Overflow system call buffers</td>
</tr>
<tr>
<td></td>
<td>A2 Exhaust the CPU resource</td>
</tr>
<tr>
<td></td>
<td>A3 Overflow the TCP/IP stack</td>
</tr>
</tbody>
</table>

4.1 Confidentiality Attacks

If an attacker, i.e., an unauthorized person, can view protected data, a successful confidentiality attack has been performed. Six such attacks are described below.

4.1.1 C1: Extract Encrypted Passwords from the SAM Database

**Synopsis**

Passwords in Windows NT are stored in the SAM database, which is accessible only by the SAM subsystem. However, if an attacker has a copy of the file containing this database, he can extract encrypted passwords that may be used later as input to a password cracking program, see Section 4.1.5.

**Attack method**

On the target system, we found that a copy of the SAM database could be expanded from the `sam._` file in the repair folder. In addition, Section 4.1.2 and Section 4.1.3 present two other ways of getting a copy of the SAM database. Once an attacker has a copy of that file, he can extract the encrypted passwords with the `SAMDump` utility, which is publicly available on the Internet.

---

\(^1\) All were successfully verified in the target environment.
Cause
A compressed version of the SAM database is stored in the $sam_.\_file$.

Attacker gain
An attacker can obtain a list of all users in the system together with their encrypted passwords.

Remedy
Windows NT has a configuration option that will make an extra level of encryption of the SAM database.

4.1.2 C2: Mount and Read an NTFS Volume

Synopsis
It was possible to boot the target PCs with a different operating system from a floppy drive and then mount the NTFS volume(s). This implies that any system that can be booted from a floppy disk is insecure.

Attack method
Download the $NTFDOS$ utility from the Internet. Reboot a target computer with MS-DOS using a boot floppy. Then mount the NTFS volume with $NTFDOS$, which is an NTFS driver for MS-DOS. The NTFS volume is now readable.

Cause
It is possible to boot most PCs by default from the floppy drive.

Attacker gain
With a fully readable NTFS volume, an attacker may, for example, read the SAM database.

Remedy
A PC must be configured not to allow booting from the floppy drive.

4.1.3 C3: Remotely Mount and Read an NTFS Volume

Synopsis
It is possible under certain circumstances to remotely mount an NTFS volume via the serial port. Any person with physical access to a Windows NT system, a (portable) computer and a null-modem cable may read the NTFS volumes.

Attack method
$NTRecover$ is a utility that can be used to recover a dead system. It consists of both client and host software, where the client program is executed on the failed system and the host
An Analysis of the Security of Windows NT

runs on the working system. The failed system is booted off a floppy directly to the NTRecover program. Once the client and the host programs are started and configured on their respective computers, native Windows NT file systems such as NTFS and FAT on the client machine are visible on the host computer.

**Cause**
It is possible to boot most PCs by default from the floppy drive.

**Attacker gain**
With a fully readable NTFS volume, an attacker may, for example, read the SAM database.

**Remedy**
Configure all computers so that they will not allow booting from the floppy drive, even though this may imply that file system errors cannot be repaired with tools such as NTRecover.

### 4.1.4 C4: Anonymous Logon

**Synopsis**
The RedButton bug makes it possible to anonymously connect to a Windows NT computer and enumerate users and shares and to read and modify some parts of the Registry.

**Attack method**
Download and execute the RedButton exploit script.

**Cause**
One of the reasons it is possible to make an anonymous logon, as far as we understand it, is as follows. In order to manage trust in a convenient way, the administrator in the trusting domain needs some way of knowing which possible users or groups shall be given rights in the trusting domain. However, since the trusted domain does not trust the trusting domain—trust is normally one-way—it needs a way of gaining access to this information. In this case, the trusting domain will open an anonymous channel to the trusted domain to obtain this information. The problem is that the anonymous user will have the rights of the everyone group, and that group has extensive privileges.

**Attacker gain**
With the RedButton program, an attacker can read sensitive information stored in the file system available to the everyone group. Furthermore, the exploit script lists all shares including the hidden ones.
Remedy
This flaw is not easily eliminated from a Windows NT system since anonymous logons are necessary for the trust mechanism to function properly, see Section 2.5.2. It is not possible simply to block the everyone group without creating a new group with almost the same privileges, and then replace every occurrence of everyone with the new group. If the everyone group is blocked, the system will not function as intended.

Comment
While Microsoft has released a hotfix and instructions on how to handle this situation, it is still dependent of a great deal of manual configuration. See Microsoft’s knowledge base article Q143474 for more information.

4.1.5 C5: Crack Passwords

Synopsis
If encrypted passwords are available, it is possible to launch a dictionary as well as a brute force attack against them. A list with encrypted passwords can either be created by the SAMDump utility described in Section 4.1.1 or through the readsmb program that is shipped together with the famous password cracker, L0phtCrack 2.0. Readsmb snatches encrypted passwords from the network.

Attack method
Download a password cracker from the Internet. L0phtCrack and Crack for Windows NT are two examples of publicly available password crackers. Then create a list of encrypted passwords as described above and, finally, let the password cracker do its task.

Cause
A publicly available password list is vulnerable to both a dictionary and brute force attack, see the comment below.

Attacker gain
The attacker can get plaintext passwords.

Remedy
In a stand-alone environment, the configuration option that will make an extra level of encryption of the SAM database can be satisfactory. In such a case, it is nearly impossible, or at least very difficult, to extract passwords from the database. In a networking environment, however, encrypted passwords will be sent over the network. These can be snatched by an attacker. An extra level of encryption, by means of using, for example, IP security (IPSec) [1, 2] for all messages sent over the network, will obstruct an attacker. Another
completely different approach is to implement a more secure authentication protocol, e.g., using smart cards.

**Comment**
The LAN Manager password is padded with NULLs if it is not 14 characters long. This together with the encryption method used creates recognizable patterns if the password contains less than eight characters. This is yet another flaw.

### 4.1.6 C6: Capture Passwords

**Synopsis**
Windows NT supports eight different authentication variants for backward compatibility reasons. The party that suggests the variant to be used is the client in the SMB_C_NEGOTIATE message [11]. The older of these variants sends plaintext passwords over the network, and some of the newer ones have weak encryption schemes. The server can always try to suggest another variant, but it is normally the client that sets the security level. By default, the server will accept plaintext passwords as valid authentication [13]. This can be used in a man-in-the-middle attack, originally suggested by David Louden, to snatch passwords directly from the network. The attack is based on the fact that the server always accepts plaintext passwords and that the client might use a weaker authentication protocol.

**Attack method**
Scan the network for negotiate request messages. When a request is intercepted, send a reply to the request masquerading as the server that is the recipient of the request and claim that you only understand an authentication variant that uses plaintext passwords. Snatch the plaintext password from the network when the client later sends the session setup message. None of the attacked parties will know that the attack has taken place, since the user on the client is not aware of the fact that the security level has been lowered and the server sees nothing wrong in the fact that the client sends plaintext passwords, even though it originally suggested an encrypted variant. A program called *c2myazz* demonstrates this weakness.

**Cause**
This flaw is caused solely by Microsoft’s ambition to be backward compatible. Thus the system accepts authentication methods with very low security.

**Attacker gain**
Plaintext passwords can be revealed by an attacker.

**Remedy**
Configure each workstation not to use older authentication variants.
4.2  **Integrity Attacks**

Unfortunately, integrity means different things in different contexts. Here, we use the word to describe the case in which data can be modified only by authorized parties. Four different attacks violating the integrity attribute will be described.

4.2.1  **I1: Mount and Update an NTFS Volume**

**Synopsis**
The possibility of booting a computer from a floppy disk and then mounting the NTFS file system not only affects the confidentiality of the system. If write access is also feasible on the mounted file system, the integrity of the system will be threatened as well.

**Attack method**
It is possible to both read and write NTFS volumes on the target computer with the ERD Commander exploit script. The exploit script actually boots a stripped down version of Windows NT.

**Cause**
The target computers can be booted from a floppy drive.

**Attacker gain**
An attacker can create, delete, or update any file in the file system, including the SAM database.

**Remedy**
Configure all computers so that they will not allow people to boot from the floppy drive.

4.2.2  **I2: Remotely Mount and Update an NTFS Volume**

**Synopsis**
File systems remotely mounted as mentioned in Section 4.1.3 are also writable, which may be used by an attacker to modify stored data.

**Attack method**
See again Section 4.1.3. This time, however, the goal is to modify data stored on the mounted volume.

**Cause**
The target computers can be booted from a floppy drive.
An Analysis of the Security of Windows NT

**Attacker gain**
An attacker can create, delete, or update any file in the file system, including the SAM database.

**Remedy**
Configure all computers so that they will not allow people to boot from the floppy drive.

4.2.3 I3: Overwrite the Administrator's Password

**Synopsis**
The *NTLocksmith* program can be used to specify a new password, i.e., it overwrites the old one, for the administrator on the system. It must be used together with *NTRecover*, see Section 4.2.2. We believe that all this program does is scan the SAM database for a certain string and then overwrite the content a certain offset from this point without knowing the real structure of the database. It is therefore highly unlikely that it will work on systems with an extra level of encryption on the SAM database.

**Attack method**
Mount the volume containing the SAM database using *NTRecover*. Launch the *NTLocksmith* utility. The new administrator password will be requested.

**Cause**
This attack was successful because the file system volume containing the SAM database was writable from another operating system.

**Attacker gain**
An attacker can overwrite the administrator’s password.

**Remedy**
Disallow people to boot from the floppy drive, which will prevent *NTRecover* from working. Another possibility is to encrypt the whole SAM database using the SYSKEY facility, introduced in SP3.

4.2.4 I4: Join the Administrator Group (*GetAdmin*)

**Synopsis**
With the *GetAdmin* exploit script, any user can join the administrator group. All users belonging to this group have local administrator privileges.

**Attack method**
Logon to a target computer. Download *GetAdmin* and execute the program as follows:

```
getadmin foo
```
After reboot, the user *foo* is a member of the administrator group.

**Cause**
This flaw is caused by a coding mistake resulting in insufficient control of parameters passed to the kernel.

**Attacker gain**
Any user can join the administrator group.

**Remedy**
Install the hotfix from Microsoft.

### 4.3 Availability Attacks

Availability means that service or information is provided to authorized users. The remainder of this section presents three different availability attacks. Another name for this type of attack is *Denial-of-Service (DoS)* attacks.

#### 4.3.1 A1: Overflow System Call Buffers

**Synopsis**
Insufficient control of parameters may lead to buffer overflow. A set of system calls with this kind of vulnerability exists in Windows NT. The *NTCrash* exploit script utilizes such flaws to crash the entire system.

**Attack method**
Download and execute the *NTCrash* exploit script. This program was primarily designed and implemented to detect and log system calls with insufficient parameter checks. Thus a list of the vulnerable system calls was also presented when the program was executed.

**Cause**
This flaw is a typical programming mistake, introduced during implementation and *not* discovered when testing the system.

**Attacker gain**
An attacker can force the system to a grinding halt and thereby deny service to others.

**Remedy**
This flaw must be patched by the vendor. To our knowledge, no such patch exists.
4.3.2 A2: Exhaust the CPU Resource

Synopsis
A user level thread in Windows NT has a priority value between 1 and 15, where 1 denotes the lowest priority. An aging mechanism that will gradually increase the priority is implemented in the system, but it will only age threads up to priority 14. The publicly available program CPUHog exploits this flaw to cause a denial of service attack.

Attack method
CPUHog demonstrates how to completely hang a target computer by setting its own priority to 15 and then entering an infinite loop.

Cause
This flaw is caused by the fact that processes run by ordinary users have the ability to set their own priority. The hotfix released by Microsoft only partially solves the problem.

Attacker gain
The flaw can be used to deny services to other users.

Remedy
Microsoft has addressed this problem in Service Pack 2 (SP2) by allowing aging up to priority level 15. This will avoid hang-up, but a degradation attack is still possible. However, not allowing a user to set the priority will eliminate the flaw.

4.3.3 A3: Overflow the TCP/IP-Stack

Synopsis
It has recently become evident that many operating systems, including Windows NT and many versions of UNIX, do not handle packet header information properly in the TCP/IP stack. This has been exploited in a number of scripts. Teardrop, bonk, boink, Land and LaTierra are all examples of such scripts. When executing one of those, the victim machine will typically either crash completely or simply freeze for a while. Microsoft has offered hotfixes for all those exploit scripts.

Attack method
Executable exploit scripts were found on the Internet. However, they were all Linux programs. These programs were therefore downloaded on computers running Linux. The rest of the attacks were straightforward. The teardrop attack, for example, was launched as follows (IP addresses changed for security reasons):

   teardrop 10.0.0.1 10.0.0.2 -t 138 -n 10
Here, the victim computer has the IP address 10.0.0.2. The flag “-t” is the port number, and the flag “-n” specifies the number of consecutive attacks. Without the hotfix, the victim computer froze and had to be rebooted. However, the hotfix from Microsoft seems to work.

**(Cause)**
In this case, the problem is that Windows NT blindly trusts the header information of received packets. This is the result of a programming mistake.

**(Attacker gain)**
The flaw can be used to remotely deny service to other users.

**(Remedy)**
Install the hotfixes released by the vendor.

## 5 Discussion and Conclusion

This section summarizes our impressions of the security of Windows NT and gives a discussion of security trends in the system. The paper is concluded at the end of this section.

### 5.1 Security Impressions

Even though the Windows NT development team had a strong ambition to make it a secure operating system, this ambition was apparently not sufficient. We have found weaknesses covering the whole “CIA” range. Perhaps the goal was set aside or diminished by the demand for backward compatibility. Windows NT also has plenty of functionality and is therefore a complex system. It is easy to overlook shortcomings and introduce flaws in such systems, but this hardly justifies missing range checks or missing tests for invalid parameters.

In some cases, Windows NT seems to rely on *security-by-obscurity*. The structure of the SAM database, for example, is not known to ordinary people. This approach may work at first, but it is probably devastating in the long run. Security must be built on concepts and methods that can be described and explained fully and still be effective.

### 5.2 Security Trends of Windows NT

In the future, we believe that the number of successful attacks against Windows NT systems will increase dramatically. We base this assumption on two facts:

First, since the source code of the Windows NT system is not easily available for the system owners, they must wait for Microsoft to correct system weaknesses. Exploits will
be spread all across the hacker community and cause a great deal of damage, usually much faster than it will take to get the fix for the problem. The only thing the system owner can do, if he does not have very good technical knowledge of the system, is to close the system in the meantime or take the risk. When the system becomes more familiar to everyone and knowledge of its function increases and spreads, other parties than Microsoft might provide fixes.

Second, the Windows NT operating system is much more homogenous than, for example, the UNIX operating system because the former had only one developer, while the latter had many. The effect of this will be that all installations of Windows NT will be affected by a newly invented attack method, since the exploited weakness is likely to be present in all of them. In a UNIX environment, on the other hand, only one variant of the operating system is usually affected, so a potential attacker must know which variant is vulnerable.

### 5.3 Conclusion

Our security analysis displays a number of weaknesses in Windows NT, and there is ample opportunity to improve the security. The relatively low number of known intrusions so far is due to the facts that the system has been fairly unknown and that the principle of security-by-obscurity has been used. However, this principle will not maintain security in the long run.

### Acknowledgments

The work described in this report was supported by Telia Research, Farsta, Sweden.
References


PAPER D

A Comparison of the Security of Windows NT and UNIX

Reprinted from

Proceedings of the Third Nordic Workshop on Secure IT Systems, NordSec ’98
Trondheim, Norway, November 5–6 1998.
A Comparison of the Security of Windows NT and UNIX

Hans Hedbom, Stefan Lindskog, Stefan Axelsson, and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
{hansh, stefanl, sax, erland.jonsson}@ce.chalmers.se

Abstract

This paper presents a brief comparison of two operating systems, Windows NT and UNIX. The comparison covers two different aspects. First we compare the main security features of the two operating systems and then make a comparison of a selection of vulnerabilities, most of which we know have been used for making real intrusions. We found that Windows NT has slightly more rigorous security features than “standard” UNIX, although the two systems display similar vulnerabilities. The differences that we might indeed experience are believed to be due to the fact that Windows NT has not been on the market as long as UNIX. Thus, the conclusion is that there is no significant difference between the “real” levels of security of these systems.

Keywords: security comparison, vulnerability, intrusion, Windows NT, UNIX.
1 Introduction

It has been claimed that the security of Windows NT is far better than that of previous commercial operating systems. In order to verify (or refute) this statement, we have made a brief comparison of the security of Windows NT to that of UNIX. UNIX was selected as a reference since it is well-known and widely spread. Thus, the target systems were (1) a networked Windows NT 4.0 and (2) UNIX with NFS (Network File System) and NIS (Network Information System). These systems constitute comparable environments, i.e., they have similar network functionality. It should however be stressed that UNIX comes in many different versions, so our reference is not completely unambiguous. Still, we believe that this fact does not reduce the value of the work presented.

The comparison covers two different aspects. We first compare the main security features of the two operating systems and then make a comparison of a selection of vulnerabilities, most of which we know have been used for making real intrusions. These were gathered from intrusion experiments carried out at the Department of Computer Engineering at Chalmers University of Technology for data collection purposes [3, 22] or from our own system analysis [15]. Some data were taken from other publicly available sources.

For the comparison of the vulnerabilities of the two systems, we used a taxonomy of intrusion techniques suggested by Lindqvist and Jonsson [14]. The taxonomy has proven useful for classifying realistic intrusions and covers all three security attributes: confidentiality, integrity and availability.

In the following, Section 2 compares the security features and Section 4 gives a systematic comparison of weaknesses in Windows NT and UNIX. The taxonomy used for classifying the vulnerabilities in Section 4 is described in Section 3. Section 5 discusses the results and concludes the paper.

2 Comparison of Security Features

The main focus of this section is on the security features provided by Windows NT and UNIX. Hence, identification, authentication, access control, and auditing are presented. The reason for choosing these mechanisms (or features) is that TCSEC [25] primarily focuses on them in the C2 requirement specification. They also represent different aspects of security and are meant to provide a broad coverage of the area. Furthermore, networking features in Windows NT and UNIX with NFS and NIS are introduced. Impersonation mechanisms in the two systems are also covered. Differences and similarities between the security mechanisms of the two operating systems are discussed in a concluding subsection. We start this section with a brief overview of the systems.
2.1 System Overviews

2.1.1 Windows NT
The Windows NT operating system was developed by Microsoft Inc. and was first released in 1992. Windows NT has support for processes, threads, symmetric multiprocessing, and distributed computing and uses an object model to manage its resources. The structure of Windows NT is a hybrid between the layered model and the client/server model [5]. The former is used in the executive, which is the only part executing in kernel mode, while the latter is used to (1) provide the user with multiple operating system environments, e.g., Windows, MS-DOS, OS/2, and POSIX1, and (2) implement parts of the operating system.

2.1.2 UNIX
The UNIX operating system was first developed in the early 1970s at AT&T Bell research laboratories. It is traditionally implemented as a single monolithic kernel that runs in kernel mode, while all user programs run in user mode. The kernel contains code for the file system and device drivers as well as for process management [1, 16]. However, UNIX has always managed large parts of many system functions, such as networking etc., outside the kernel, in user mode processes.

2.2 Identification
An operating system bases much of its security on knowing who a user of the system is. The system thus requires some identification before it grants resources to a person.

2.2.1 Windows NT
User identification is done by a username that is mapped onto an internal Security IDentifier (SID). A SID is a numeric value that is unique within a domain (as well as with high probability between domains). When a new account is created on the system, a SID is created and stored together with this account. SIDs are never reused, so a newly created account can not get the SID of a previously deleted account.

2.2.2 UNIX
A user is identified by a username, which is given when the user logs on to the system. Internally, a user is identified with a User IDentification number (UID), which is a numeric value selected by the system administrator at the time the account is created. In most cases, selecting unique UIDs for each user is a good idea [6], though not strictly

---

1 Portable Operating System Interface based on uniX
required. The mapping of username to UID is kept in the file /etc/passwd, but is today often centrally managed by NIS. The super user (or root) has UID 0 (zero). Every user belongs to one or more groups. A group is identified with a group identification number (GID).

2.3 Authentication

After identification, the user must prove his identity. This process is called authentication. Most authentication systems are based on a secret shared only by the system and the user. In both Windows NT and UNIX, the authentication mechanism is a password, i.e., a secret word known to the system and the user.

2.3.1 Windows NT

User authentication is done by means of passwords. Windows NT stores passwords together with SIDs and other information about the user in a database handled by the Security Accounts Manager (SAM) subsystem. Two hashed versions of the password are stored, Lan Manager-hash (LM-hash) and NT-native, unless the system is told to use just one. The NT-native variant is stored using MD4 and the LM-hash using a variant of DES.

2.3.2 UNIX

After identification, UNIX will also request a password to authenticate the user's identity. When the user has entered the password, it is encrypted using a modified DES algorithm described in [6] and compared with the encrypted password stored in /etc/passwd (or the NIS database). If the two match, the user has been proven to be a legitimate user in the system. The file /etc/passwd is readable for everyone in the system, which makes it sensitive to password attacks. A solution to this problem is to use what is known as a "shadow" file (/etc/shadow). The whole idea is then to move the encrypted passwords from /etc/passwd to /etc/shadow and make the latter not readable to normal users.

2.4 Access Control

Access to resources must be protected. Only authorized users, or processes, should be granted access to a resource. This protection mechanism is referred to as the access control. Both Windows NT and UNIX implement discretionary access control, which means that the resource owner specifies who may access it and how.

2.4.1 Windows NT

An Access Control List (ACL) is associated with every object in the system. This list consists of a number of Access Control Entries (ACE). Every ACE is associated with a user
(or a group) SID and holds the actions that this user is allowed or disallowed to perform on this object. ACEs that disallow are put before ACEs that allow in the ACL. A user that does not have an ACE in the ACL has no access at all to that object.

An object can also have a NULL ACL or an empty ACL. If the object has a NULL ACL this object has no restrictions. An empty ACL on the other hand means that none can access this object in any way. A newly created object is usually given the ACL of its creator by default.

When a user is authenticated to the system, a token is created for this user. This token is called the primary token. It contains, among other things, the SID for the user and the SIDs of the groups in which this user is a member. This token is compared with an object’s ACL to grant (or deny) the user access to this object.

2.4.2 UNIX

UNIX’s access control is implemented through the file system. Each file (or directory) has a number of attributes, including a filename, permission bits, a UID and a GID. The UID of a file specifies its owner.

The permission bits are used to specify permissions to read (r), write (w) and execute (x) the file for the user, for the members of the user’s group, and for all other users in the system. Permissions rwxr-xr-x specify that the owner may read, write and execute the file, the group members may read and execute it, and all others may only execute the file. A dash (“-”) in the permission set indicates that the access rights are disallowed. Most systems today also support some form of ACL schemes.

Furthermore, each process in UNIX has an effective and a real UID and an effective and a real GID that are associated with it. Whenever a process attempts to access a file, the kernel will use the processes’ effective UID and GID to compare them with the UID and GID associated with the file in order to decide whether to grant the request.

2.5 Auditing

The process of monitoring the activity in a system is known as auditing. The purpose of auditing is twofold: first, to verify that the protection mechanisms actually work as intended, and, second, to keep track of what has happened in the system.

2.5.1 Windows NT

The Security Reference Monitor (SRM) and the Local Security Authority (LSA), together with the Event Logger, handle auditing in Windows NT. Different types of events are grouped into event categories, and auditing is then done on the basis of these groups. There are seven types of event groups. For details on event groups, see [18]. The audit policy determines whether auditing is applicable and, if so, decides what is to be recorded.
The policy is managed by the LSA, which hands it over to the SRM, which in turn is responsible for its implementation.

The auditing is based on audit records constructed on the request of the server responsible by the SRM (in some cases by the LSA). Requests from the executive are always carried out. Servers, on the other hand, need the audit privilege for their requests to be honoured. The request must be sent for each occurrence of an event. The audit record is then sent to the LSA, which in turn sends it to the Event Logger after it has expanded certain fields and compressed others. The Event Logger commits the audit record to permanent storage.

2.5.2 UNIX

Traditionally, the UNIX kernel and system processes store pertinent information in log files, either locally or centrally on a network server, via the flexible and configurable syslog facility [12]. In addition, many modern UNIX systems support a more comprehensive type of auditing known as C2 audit. This is so named because it fulfils the audit requirements for the TCSEC C2 security level [25].

2.6 Networking

Most current computer systems are networked. So are also Windows NT and most UNIX systems. Some network services offered by the two systems are briefly discussed below.

2.6.1 Windows NT

The distributed parts of Windows NT rely heavily on Server Message Blocks (SMBs). This is an application level protocol used by Microsoft for a number of things, among them authentication, RPC, and the Common Internet File System protocol (CIFS) [7, 10, 11]. In the Windows NT environment, an SMB is carried on top of a NetBIOS over a TCP/IP (NBT) session [19, 20], including UDP as a carrier for NetBIOS as well. There are a number of things to be said about CIFS/SMB, as regards security. First, it is possible to establish so-called NULL sessions, i.e., sessions with no username. Machines that exchange RPCs through named pipes frequently do this. Second, all logging in Windows NT and most other checks are done on computer names and not IP addresses. This means that there is no way of telling the location from which a computer is accessing. We are fully aware of IP address forging, but it is even simpler to forge a computer name. Third, the protocol is very outspoken and will freely give away much information about what is going on. Last, all in all, there is too much trust in client machines behaving in a non-malicious manner. For a more in-depth description of some of the weaknesses in CIFS/SMB, see for example [9].
2.6.2 UNIX

Most modern UNIX machines are networked. The default networking support is based on TCP/IP. Remote terminal access, remote file transfer and remote command execution are provided through a set of programs (rlogin, rcp, rsh, etc.) collectively known as the ‘r’ commands. The Network File System (NFS) [23] adds support for several hosts to share files over the network, while the Network Information System (NIS) [8], formally known as the Sun Yellow Pages, allows hosts to share system databases containing data concerning user account information, group membership, mail aliases etc. via the network, to facilitate centralised administration of the system.

The ‘r’ commands are not secure, for a number of reasons. First, they assume that all hosts in the network are trusted to play by the rules, e.g., any request coming from a TCP/IP port below 1024 is considered to be trusted. Second, they use address-based authentication, i.e., the source address of a request is used to decide whether to grant a service. Third, they send cleartext passwords over the network.

Before an NFS client can access files on a file system exported by an NFS server, it must mount the file system. If a mount operation succeeds, the server will respond with a “file handle”, which is later used in all accesses to that file system in order to verify that the request is coming from a legitimate client. Only clients that are trusted by the server are allowed to mount a file system. The primary problem with NFS is the weak authentication of the mount request [4], which is based on IP addresses that may be faked.

NIS can be configured to perform (1) no authentication, (2) traditional UNIX authentication based on machine identification and UID, or (3) DES authentication [8]. Method (3) provides quite strong security, while (2) is used by default by NFS. According to [6], a fourth authentication method based on Kerberos [24] is also supported by NIS. Both servers and clients are sensitive to attacks, though Hess et al. [8] are of the opinion that the real security problem in NIS resides on the client side. It is easy for an intruder to fake a reply from the NIS server. Such an intrusion is further described in section 4.1.3.2.

2.7 Impersonation

The user of the system must be able to perform certain security critical functions on the system which he has normally not the right to do. One way of solving this problem is to give the user controlled access to a limited set of system privileges to allow the execution of a specified process with system privileges. This specified process can then perform application level checks to ensure that the process does not perform actions that the user was not intended to be able to perform. This of course places stringent requirements on the process in terms of correctness of execution, lest the user be able to circumvent the security checks and perform arbitrary actions with system privileges.

1 “He” should be read as “he or she” throughout this paper.
2.7.1 Windows NT
Every thread that executes in the system has the possibility of having two tokens, one the primary token and the other a so-called impersonation token. This is a token given to the thread by another subject which allows the thread to act on that subject’s behalf. The impersonation token is a full or restricted variant of that subject’s primary token.

2.7.2 UNIX
Two separate but similar mechanisms handle impersonation in UNIX, the so-called set-UID (SUID) and set-GID (SGID) mechanisms. Every executable file on a file system so configured can be marked for SUID/SGID execution. Such a file is executed with the permissions of the owner/group of the file instead of that of the current user. Typically, certain services that require super user privileges are wrapped in a SUID super user program, and the users of the system are given permission to execute this program. If the program can be subverted into performing some action that it was not originally intended to perform, serious breaches of security can result.

2.8 Discussion of Security Features
There are similarities in the security features of the two systems. Our opinion is, however, that the Windows NT system’s mechanisms are more ambitious than the standard UNIX mechanisms. This is of course partly due to the fact that Windows NT is of a later date than UNIX. However, most Windows NT security features are available for modern UNIX systems.

It is interesting to note that both systems contain mechanisms for impersonation, while this has been an almost endless source of security problems with UNIX. It will be interesting to see whether Windows NT will come to suffer the same problems.

Finally, it is worth mentioning that a system need not be more secure than another just because it has more and better security features. Unfortunately, in both Windows NT and UNIX, most of these features are disabled by default for reasons of convenience. Thus, it requires an active decision to introduce security during installation.

3 Classification of Computer Security Weaknesses
A taxonomy is used to categorize phenomena, which, in general, make system studies possible or at least easier. By using an established classification scheme of intrusion techniques, different systems can be compared on the basis of intrusion data. One such scheme was suggested by Neumann and Parker in [21] and was refined by Lindqvist and Jonsson [14]. We chose the latter in our comparison.
The selected taxonomy consists of three categories: bypassing intended controls (NP5), active misuse of resources (NP6) and passive misuse of resources (NP7), see Table 1.

<table>
<thead>
<tr>
<th>Category</th>
<th>Bypassing Intended Controls (NP5)</th>
<th>Active Misuse of Resources (NP6)</th>
<th>Passive Misuse of Resources (NP7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password Attacks</td>
<td>Capture</td>
<td>Exploiting Inadvertent Write Permissions</td>
<td>Manual Browsing</td>
</tr>
<tr>
<td>Spoofing Privileged Programs</td>
<td>Guessing</td>
<td>Resource Exhaustion</td>
<td>Automated Searching</td>
</tr>
<tr>
<td>Utilizing Weak Authentication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bypassing Intended Controls (NP5).** The act of circumventing mechanisms in the system that are put there to stop users from performing unauthorized actions.

**Active Misuse of Resources (NP6).** The act of maliciously using permissions or resources to which the user accidentally gains authorization.

**Passive Misuse of Resources (NP7).** The act of searching for weak or erroneous configurations in the system without having authorization to do so.

Descriptions of the applicable subclasses of each category are further presented in Section 4. Other taxonomies exist as well, though none, including the two mentioned above, is perfect. A survey of previous work in the field is presented in [14].

# 4 Systematic Comparison of Weaknesses

This section gives examples of security weaknesses in the Windows NT system and the UNIX system. The different attacks are categorized according to the taxonomy presented in Section 3. Vulnerabilities in the same class are compared with each other. In the cases in which the attacks are identical in both systems, only one example is given and discussed.

## 4.1 Bypassing Intended Controls (NP5)

This category consists of three subclasses: password attacks, spoofing privileged programs, and utilizing weak authentication. The first subclass, password attacks, is further
subdivided into password guessing and password capturing. Examples from each of those subclasses are given below.

4.1.1 Password Attacks

To learn the password for a certain account, the attacker can either capture a plaintext password or a cryptographically hashed password. In the latter case, the attacker can then attempt to either brute force the password, i.e., try all possible combinations, or with greater chance of speedy success, try guessing likely passwords, e.g., those based on the name of the user.

4.1.1.1 Password Capturing Attacks in Windows NT

Windows NT supports eight different variants of authentication for reasons of backward compatibility. The party that suggests which variant should be used is the client in the SMB_C_NEGOTIATE message. The older of these variants sends plaintext passwords over the network. The server can always try to suggest a more secure variant, but it is the client that, unless there is a share-level versus user-level incompatibility, sets the level of security. The server as default will always accept plaintext passwords as valid authentication [9]. An attack suggested by David Loudon, that is based on the fact that the server always accepts plaintext passwords and that the client can be fooled into using a weaker authentication protocol, is carried out as follows:

Scan the network for negotiate request messages. When a request is intercepted, send a reply to that request masquerading as the server that is the recipient of the request and claim that you only understand an authentication variant that uses plaintext passwords. Snatch the plaintext password from the network when the client later sends the session setup message. None of the attacked parties will know that the attack has taken place, since the user on the client is not aware of the fact that the security has been lowered, and the server sees nothing wrong in the fact that the client sends plaintext passwords, even though it originally suggested an encrypted variant.

4.1.1.2 Password Capturing Attacks in UNIX

Some of the protocols, e.g., telnet, ftp, and the ‘r’ commands often used in UNIX environments, send plaintext password information. Thus all an attacker has to do is sniff the network for these activities and copy the password off the wire.

4.1.1.3 Comparison of Password Capturing Attacks

The weaknesses described here can all be categorized as legacy problems. Windows NT has this problem because it tries to be backward compatible, and the same may be said about UNIX environments that uses the protocols described above. If compatibility is not a requirement, a Windows NT Server or Client can be configured not to accept older vari-
ants of the SMB protocol and, in a UNIX environment, more secure protocols that have the same functionality, e.g., SSH [26], can be used.

4.1.1.4 Password Guessing Attacks in Windows NT
In Windows NT, the passwords are stored in Lan-Manager-hash, or LM-hash, format as well as in NT-native format. The LM format has a much weaker encryption algorithm than the native format, e.g., in Lan Manager, only uppercase characters are allowed and the password is padded with NULLs if it is not 14 characters long. This, together with the encryption method used, creates recognizable patterns if the password contains less than eight characters. The public domain program L0phtCrack can be used for brute force attacks and for dictionary attacks. To gather encrypted passwords for later cracking, one can either try to get hold of a copy of the SAM database or sniff the network. Programs exist for both purposes.

4.1.1.5 Password Guessing Attacks in UNIX
In UNIX, encrypted passwords are traditionally stored in the /etc/passwd file. This file is readable for every legitimate user on the system. In some UNIX versions, the encrypted passwords are instead stored in a shadow file, which is readable only by the super user. Since the introduction of NIS, it is also possible to obtain passwords from the network in UNIX installations. NIS does very little to prevent unauthorized hosts from gaining access to the hashed passwords of a system. When these passwords have been obtained, the attacker is free to try to crack them with, for example, Crack [17].

4.1.1.6 Comparison of Password Guessing Attacks
Both Windows NT and UNIX are susceptible to password attacks, not actually because of weak encryption methods but due to the fact that they allow too short or easily guessed passwords. An alternative way of looking at this is to say that they allow passwords to have a longer lifetime than the time it takes to guess or break them.

4.1.2 Spoofing Privileged Programs
If a program that is more trusted by the system than is a particular user, can be lured to perform actions on behalf of that user, he will have access to more privileges than intended. The program can be fooled into giving away information, changing information or causing denial of service attacks. There are a number of examples of exploits in this category, including GetAdmin [15] and the X terminal emulator logging vulnerability [13]. Below we present a vulnerability that is identical in both systems.

4.1.2.1 Spoofing Privileged Programs in Windows NT and UNIX
When an IP packet handler receives a message that is too large to fit into a packet, it normally breaks this message into fragments and puts every fragment in a separate IP packet.
All but the last of these packets have the *More Fragments* (MF) bit set in the header. In addition, these packets have a *Fragment offset* that specifies where in the original message this fragment belongs.

*Teardrop* is an attack, or more correctly a program, that exploits the fragmentation handling of the IP stack. The idea is to send a 2-fragment IP packet, with one fragment too small. This results in a negative data length value, which causes the IP stack to copy too much data and crash. This is a typical so-called buffer overrun. Windows NT and many UNIX versions, as well as routers, are sensitive to *Teardrop* attacks, much because the same reference implementation of the TCP/IP stack has been used in all systems.

### 4.1.3 Utilizing Weak Authentication

Utilizing weak authentication means taking advantage of the fact that the system does not properly authenticate the originator of a certain request. In the paragraphs below, we will exemplify this class by presenting two man-in-the-middle attacks.

#### 4.1.3.1 Utilizing Weak Authentication in Windows NT

In Windows NT, a normal remote logon occurs as follows:

1. The client tries to set up an SMB connection to the exported service on the remote computer.
2. The server will send the client a challenge.
3. The client will calculate two 24-byte strings using the challenge and the LM and NT passwords, and send those in an SMB message to the remote computer. The user is then considered authenticated.

The protocol can be fooled, however, as the following attack shows. It is described by Brezinski in [2] and relies on the fact that there is nothing that prevents an attacker from masquerading as the server. Brezinski describes the attack as follows (Mallory is the attacker and Alice the user):

1. Mallory sends a connection request to the server.
2. The server responds with a challenge, i.e., a random string.
3. Mallory waits for Alice to send a connection request to the server.
4. When Alice sends a connection request to the server, Mallory forges a response to Alice containing the challenge sent to Mallory by the server.
5. Alice encrypts the random string using the hash of her password as the key and

---

1 The length value will be interpreted as a very large positive value, since it is represented as an unsigned.
sends it to the server.

6. Mallory intercepts (or simply copies it off the wire) Alice’s response and repackages it as a response to the connection request made in 1 and sends it to the server, claiming to be Alice.

7. The server looks up the hash of Alice's password in the security database and encrypts the random string sent to Mallory.

If the server's encrypted string matches the encrypted string sent by Mallory, claiming to be Alice, to the server, Mallory is allowed into the system under Alice’s credentials.

4.1.3.2 Utilizing Weak Authentication in UNIX

When NIS is added to the basic operating system, similar attacks are possible in UNIX as well. One described by Hess et al. in [8] goes as follows:

1. An intruder is watching on the connection between the NIS client and the NIS server for the NIS/RPC/UDP yp_match query for a particular user. This command is used to obtain authentication information from a NIS server. The reply is a string identical to a user entry in the /etc/passwd file.

2. When this query passes by the intruder, it quickly generates a fake reply and sends this to the client before the server sends its reply. Since UDP is used and the server’s reply is later than the intruder’s, the latter message is simply discarded.

The result of this attack is that the user is authenticated by the intruder and not by the proper NIS server.

4.1.3.3 Comparison

These two attacks succeed because of misplaced trust. In both cases, the client trusts the server blindly and, because of that, it can be fooled. The results differ slightly. In the Windows NT case, the intruder gains access to the server and only as the user that he manages to intercept. In the UNIX case, on the other hand, the intruder gains access to the client but as a user of his own choice. One can, however, easily think of variants in which the tables are turned in both cases.

4.2 Active Misuse of Resources (NP6)

Active misuse of resources is divided into two subclasses: exploiting inadvertent write permissions and resource exhaustion. Weaknesses in the first subclass can usually be traced to configuration flaws and are therefore very local in nature. For this reason, we have chosen only to present examples from the last category.
4.2.1 Resource Exhaustion

Resource exhaustion is usually employed as a means of causing denial of service. The idea is to allocate as many instances of one (or more) type(s) of resources as possible. Normally, this will either slow down or crash the system.

4.2.1.1 Resource Exhaustion in Windows NT

A thread in Windows NT usually has a priority value between 1 and 15, where 1 is the lowest priority. It is not normal for a program to have a high priority value (>10). Furthermore, Windows NT has an aging mechanism that will give threads higher priority. However, this mechanism will only age threads up to priority 14. CpuHog is a small program that uses the priority mechanism to hang the system. What CpuHog actually does is to set priority 15 on itself and then enter an infinite loop. This will cause the system to hang so that it is impossible to start any other program, including the Task Manager. The strange thing here is that you need no special privileges to be able to do this. Microsoft has addressed this problem in a service pack by allowing aging up to priority level 15, which means that the effect of CpuHog is only to slow down the system considerably.

4.2.1.2 Resource Exhaustion in UNIX

Probably the most known denial of service attack in the UNIX environment is the while (1) fork(); program. This line of C code starts a new process for every iteration in the while loop. The result will be that all entries in the system process table are consumed, which implies that no new processes can start. Many vendors have now fixed this problem by limiting the number of processes that a user can be started.

4.2.1.3 Comparison

Resource exhaustion attacks are possible in both systems, due to the fact that they allow users to allocate an unlimited number of resources. It is interesting to note that the two attacks use different mechanisms, but that the result is the same.

4.3 Passive Misuse of Resources (NP7)

Passive misuse of resources is the idea of an unauthorized user looking for weaknesses without using them. This knowledge can later be used in an active attack. The methods used are either manual browsing or automated browsing with the use of specialized tools.

4.3.1 Manual Browsing

In manual browsing, the attacker looks for weaknesses without using any tool designed for this purpose. The knowledge gained by manual browsing can later be incorporated into an automatic tool. It is difficult to say whether one of the two systems is more susceptible to manual browsing attacks than the other. The key thing here is a good understanding of the
A Comparison of the Security of Windows NT and UNIX

4.3.2 Automated Searching

Automated searching is subdivided into using a personal tool or using a publicly available tool. The only reason for this division is that it is easier to automatically detect the use of a publicly available tool.

Many different tools that look for weaknesses or other information exist for both environments. In some cases, these tools have the same name and look for both general and operating system specific vulnerabilities. A program in this category is the Internet Scanner (IS), which is a commercial version of the well known program Internet Security Scanner (ISS). Other general programs that have long been used in the UNIX community either will be ported or are now in the process of being ported to Windows NT. SATAN is an example of such a program. All in all, both systems are definitely sensitive to these kinds of attacks.

5 Discussion and Conclusion

This paper demonstrates that the security mechanisms of Windows NT are slightly better than those of UNIX. Despite this fact, the two systems display a similar set of vulnerabilities. This implies that Windows NT has the theoretical capacity of being more secure than “standard” UNIX. However, with the present way of installing and using the systems, there seems to be no significant difference between their security levels. It is true that there are presently more intrusions in UNIX systems, but we believe that this is due to the aging factor, i.e., the statement above should hold when comparing the systems at the same state of development and market penetration. Thus, the only reason for more UNIX penetrations is that the system is older and better known, and we should anticipate an increasing number of intrusions into Windows NT, a tendency that has already started to be observed.

It is clear that the Achilles heel of both systems is networking, and both systems utilize the same low level protocols, i.e., IP, TCP and UDP, and comparable high level protocols. This may, to some extent, explain why the security behaviour of both systems is similar, but it does not provide a full explanation. However, as long as networking is such a weak point, the usefulness of other security mechanisms is diminished.
Acknowledgments

The work described in this paper was partly supported by Telia Research, Farsta, Sweden.
References


PAPER E

Security Evolution of a Network Operating System

Reprinted from

Proceedings of the Fourth Nordic Workshop on
Secure IT systems, NordSec ’99
Security Evolution of a Network Operating System

Stefan Lindskog\(^1\), Ulf Gustafson\(^2\), and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
{stefanl, ulfg, erland.jonsson}@ce.chalmers.se

Abstract

In this paper, we study the security evolution of Novell NetWare from version 3.12 until the most recent current version, i.e., version 5. A detailed description of both security features and flaws in Netware 3.12 is presented. The flaws were revealed during an intrusion experiment performed by undergraduate students. The paper also deals with new security features, as well as newly discovered weaknesses, in versions 4 and 5 of the network operating system. The paper concludes that the weakest points in the Novell NetWare system are the clients themselves and the connections between clients and servers. Surprisingly enough, this remains true despite significant security improvements in later versions of the system. It is discussed whether this result can be generalized, leading to the conclusion that virtually no commercial system will ever be secure.

Keywords: security evolution, analysis, vulnerability, intrusion, Novell NetWare.

\(^1\) The author is also with the Department of Computer Science, Karlstad University, SE-651 88 Karlstad, Sweden.

\(^2\) Author’s present address: Ericsson Mobile Data Design AB, S:t Sigfridsgatan 89, SE-412 66 Göteborg, Sweden.
1 Introduction

During the 1990s, many systems were marketed as offering a high degree of security or at least as being security enhanced as compared with previous versions. In the present paper, we therefore study the security evolution of Novell NetWare from version 3.12 to version 5. The results of this paper are based primarily on vulnerabilities revealed during the intrusion experiment described below, but also on other information on system weaknesses. Information on security mechanisms is taken from the textbooks [4, 18].

In an intrusion experiment at our Department, a number of security flaws were discovered in Novell NetWare 3.12. Final year Master of Science students in Computer Science and Engineering adopted the role of attackers. In all, 14 students were involved and worked in groups of two. We expected each group to spend approximately 40 hours of effective working time. The experiment is further described in [7], and an interpretation of the intrusion data collected is presented in [6]. Four similar intrusion experiments, in which the target system was a networked UNIX system, have also been conducted [3, 11, 17]. In addition, penetration testing [1, 5, 8] has been used in two system analyses [13, 14].

The present paper focuses on the security evolution of Novell’s NetWare operating system. We start by giving an introduction to the system and its features, followed by a description of a number of security flaws of Novell NetWare, version 3.12. It is then discussed to what extent security enhancements of later versions can remedy the vulnerabilities in version 3.12. It is found that security has been improved, but there are still severe remaining weaknesses in the system.

In the following, Section 2 presents a system overview of Novell NetWare 3.12, while Section 3 focuses on the security features provided. Section 4 gives a detailed description of the security flaws found during the experiment. Section 5 presents security improvements in Novell NetWare 4 and 5 and gives some hints about newly discovered flaws in those versions of the operating system. Section 6 concludes the paper.

2 System Overview of NetWare 3.12

NetWare 3.12 is a network operating system that allows many different clients, such as PCs running MS-DOS, Windows or OS/2 as well as Macintosh and UNIX systems, to connect to a NetWare file server. Packets transmitted over the NetWare network normally use Novell’s own Protocol SPX/IPX. Since our target system in the experiment conducted consisted entirely of PCs, the following discussion is simplified by focusing only on such systems.
The clients and their server interact according to the general principles shown in Figure 1. Clients in the target system were ordinary PCs running MS-DOS 6.2 and Windows 3.1, although two additional software packages, NETX and IPX, were needed to communicate with the NetWare server (also PC-based). The NETX package redirects application system calls either to the local operating system or to the server accessible through IPX and the network interface card (NIC).

The server runs the network operating system, NetWare OS. NetWare OS can be configured and extended with various NetWare Loadable Modules, NLMs. Examples of NLMs are device drivers for network cards and hard disks, but they can also be various menu-driven application programs such as commands and programs available at the server console. When NetWare OS is running, it is not possible to start MS-DOS sessions.

### 3 Security Features in NetWare 3.12

NetWare 3.12 offers a broad range of security features. Unfortunately, most of them are not activated by default. Instead, some default values must be changed by the system administrator to attain an adequate degree of security. The remainder of this section describes users and groups, file system security, login security, account security, packet signatures, and auditing.
3.1 Users and Groups

Resources on the file server are available only to legitimate users, i.e., a user with a valid user account. GUEST and SUPERVISOR are so-called default accounts. They are automatically created when NetWare is installed on a server. The SUPERVISOR account has full rights and power on the server, and these rights cannot be taken away! Beyond these accounts, the supervisor may create other accounts if needed.

A set of users that needs similar rights can be arranged in a group. With such an arrangement, rights can be granted (or revoked) in a single operation to all members of a particular group. The EVERYONE group is automatically created when NetWare is installed. Both default accounts are members of this group. New groups may be created if needed, and a user may belong to many groups at the same time.

The server keeps track of users and groups in the bindery database, which is further described in Section 3.4.1.

3.2 File System Security

The NetWare server’s primary task is to provide file service to its connected clients, and each file on the server must only be accessible by authorized users. The latter is ensured through the file system security mechanism, which consists of two separate parts: rights security and attribute security.

3.2.1 Rights Security

Rights security is the specified access permission on a file or a directory. It is implemented through Access Control Lists (ACLs) [19]. File, as well as directory, permissions can be assigned per user or per group. Eight different rights can be specified for each file or directory: Supervisor (S), Read (R), Write (W), Create (C), Erase (E), Modify (M), File scan (F), and Access control (A). Rights security is a rather complicated mechanism and uses two functions called trustee assignments and inherited right masks to determine the user’s rights to a file. A user’s rights are the rights given to the user directly, plus the rights given to the user by group assignments and rights given to the user by inherited rights masks. When users and groups are explicitly granted rights to files and directories, trustee assignments are made. Inherited right masks, on the other hand, are rights inherited to a file or directory below a given directory with original trustee assignment. Hence, a user with, say, read permission on a directory will automatically have the same permissions on all underlying sub-directories in the directory tree [4].

3.2.2 Attribute Security

Attribute security offers users the possibility to assign attributes to files or directories, provided the user has the Modify (M) right on the file (or directory). Assigning attributes to,
for example, a file will influence all users allowed to access that specific file. There are attributes for hiding and for denial of deleting or copying a file. All in all, there are 14 different file attributes, and five of them are also applicable on directories. Similar attributes exist in the MS-DOS operating system as well. In fact, the four different file and directory attributes in MS-DOS are included in Netware.

3.3 Login Security

The login security process decides whether a user on a client PC should have access to resources on the server. The login process offers both identification and authentication. Identification is established by sending a unique user identifier to the server, which responds with an account identifier and a unique encryption key. The client then encrypts the user password and the account identifier with the key and sends it back to the server [18]. If the password and the user identifier match the corresponding pair in the bindery database, see Section 3.4.1, the user is authenticated, i.e., the user has been proven to be a legitimate user in the system.

3.4 Account Security

Account information must be securely stored somewhere in the system. In NetWare, the bindery database is used for that purpose. Furthermore, with account restrictions and intruder detection and lockout, additional security related to accounts can be achieved. These three features are further described below.

3.4.1 The Bindery Database

The bindery database is the NetWare OS system database, and every NetWare 3.12 server has its own bindery database. It is stored on the server and contains information about all users, groups, print queues etc. on the server. It is of importance that only the supervisor, the account managers, and the workgroup managers have the ability to change information in this database. Account managers and workgroup managers are users with special privileges. These extra privileges have been delegated to them by the supervisor.

3.4.2 Account Restrictions

A number of restrictions may be placed on a user’s account. Such restrictions are specified when the account is created, although they may later be changed by the supervisor, an account manager, or a workgroup manager. Below we summarize the restrictions that may be specified for an account.

- Account disabled. If this option is set, it is impossible to login to that account.
– **Account has expiration date.** With this option, an account expiration date can be specified.

– **Limit concurrent connections.** This alternative specifies how many simultaneously concurrent connections a user is allowed to have.

– **Allow user to change password.** With this restriction set, the account password cannot be changed.

– **Require password.** If this option is set, the account is required to have a password.

– **Minimal password length.** This alternative sets limitations on the minimum number of characters of which a password must consist.

– **Force periodic password changes.** This choice forces a password change at regular intervals.

– **Number of grace logins.** The number of grace logins is the number of times a user can log in after the password has expired.

– **Require unique passwords.** When this feature is activated, NetWare records the last eight passwords, and the user will not be allowed to reuse any of them.

### 3.4.3 Intruder Detection and Lockout

Another security feature related to accounts is *intruder detection and lockout*. When this feature is activated, incorrect login attempts within a specified time span are recorded. If the number of incorrect passwords exceeds a predefined threshold, the account becomes locked for a specified period of time, although the superuser may unlock it. If the SUPERVISOR accounts become locked, there is a special way of unlocking this account. The supervisor, or someone else, can always enable logins at the file server console!

### 3.5 NCP Packet Signatures

The NetWare system is based upon a client/server model. To support this model, Novell developed a protocol called NetWare Core Protocol (NCP), which essentially consists of hundreds of Remote Procedure Calls (RPCs) [2]. Unfortunately, it is not difficult to forge NCP packets transmitted over the network. Novell thus offers a security feature called *NCP packet signatures*. This feature prevents, according to Novell’s documentation, packet forgery by requiring the client and the server to sign every packet with a signature.

There are four levels (0-3) of packet signatures, which can be set on both the client and the server side. Level 0 specifies that packet signatures are not used at all, while level 1

---

1 Note that the SUPERVISOR account cannot be disabled or given an expiration date.
specifies that packet signatures will be used if requested by the other side. If level 2 is used, packet signatures will be used if the other side will allow it. Finally, level 3 is used if packet signatures are required. The default packet signature level is 1.

3.6 Auditing

To our knowledge, Netware 3.12 does not offer any kind of file system auditing or logging facility. In [4], the author mentions two security attributes, see Section 3.2.2, which are intended to be used for that purpose: Read audit (Ra) and Write audit (Wa). On page 85, he points out that “These attributes have no function at present, but may in the future be used in conjunction with an Audit Trail System”. With Ra and Wa, it would be possible to track both read and write accesses to a file, but they are not applicable to directories. However, there is a facility called accounting which may be used to track logins, although this feature is not activated by default.

4 Security Flaws in NetWare 3.12

This section presents a detailed description of security flaws found during the intrusion experiment. The target system was a “standard” Novell NetWare 3.12 system with eight PCs connected to a file server. All PCs were installed with MS-DOS 6.2 and Windows 3.1. The server was physically secured, and only the system administrator had access to it. The intruder detection and lockout feature described in Section 3.4.3 was enabled, and the number of login attempts was restricted to ten. Each group was given an account with ordinary user privileges. NCP packet signatures, see Section 3.5, were not used. The groups performed a number of successful intrusions. Five of these, all of which represent publicly known attack methods, are described below, see Table 1.

The attack methods are classified into three categories: confidentiality attacks, integrity attacks, and availability attacks. A confidentiality attack is performed when an unauthorized user manages to read protected data, while an integrity attack is the case in which an unauthorized user manages either to modify protected data or change the system configuration. If an attacker manages to force a service to be denied to authorized users, an availability attack is said to have been performed. In some cases, it is not obvious in which category a given attack should be classified. For example, assume an attack in which a plaintext password is captured from the network. This is a typical confidentiality attack. However, the password revealed may later be used in either (or both) an integrity or an availability attack. This illustrates the complexity of an intrusion process and the problem of making a distinction between initial attack, intrusion propagation, and intrusion result. This topic is further discussed in [12]. Our categorization is made with respect to the ini-
tial attack. Hence, the password capturing attack described above would have been classified as a confidentiality attack.

**Table 1. Attack methods**

<table>
<thead>
<tr>
<th>Category</th>
<th>Attack Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality attacks</td>
<td>C1</td>
</tr>
<tr>
<td>Integrity attacks</td>
<td>I1</td>
</tr>
<tr>
<td></td>
<td>I2</td>
</tr>
<tr>
<td></td>
<td>I3</td>
</tr>
<tr>
<td></td>
<td>I4</td>
</tr>
</tbody>
</table>

### 4.1 Confidentiality Attacks

#### 4.1.1 C1: Snoop Network Traffic

**Synopsis**

The target system used a network setup in which all messages except passwords were sent in plaintext. Thus, it was possible to monitor the contents of the transmitted packets.

**Attack method**

Some groups downloaded and tried network snooping programs. However, the passwords were encrypted, and the data given by the snooping programs were not very useful, mostly because of a lack of ordinary users in the target environment but also because of very complicated user interfaces.

**Cause**

Everything sent over the network was, except NetWare passwords, in plaintext.

**Attacker gain**

Secret information, such as credit card numbers and non-Netware passwords, e.g., ftp and telnet passwords, may be revealed by an attacker.

**Remedy**

The encryption of packets sent over the network would obstruct an attacker in performing this kind of attack. Unfortunately, such functionality was not offered in the target environment.
4.2 Integrity Attacks

4.2.1 I1: Remotely Controlling Client PCs

Synopsis
A PC configured with MS-DOS and the NetBIOS protocol could be configured in such a way that it was possible to remotely control it from another computer.

Attack method
A publicly available utility called spy can be used for this purpose. It essentially contains a set of programs of which two are of special importance: spytstr, which is installed on the target computer, and spy, which is installed on the attacker’s computer. The former is a so-called TSR (Terminate and Stay Resident) program. Such a program is normally started when the computer boots or when another program is launched. When started, it stays resident in the memory and performs its intended hidden task, e.g., sends keystrokes or screen contents to spy on the attacker’s computer.

Cause
This flaw is caused by the fact that TSR programs can be installed by anyone with physical access to a PC in the target environment.

Attacker gain
An attacker can remotely snoop keystrokes, read everything that is presented on the screen, change data or the setup on the target PC, and restart it.

Remedy
Either boot the clients via the network, as suggested in [15], or boot from a write-protected or encrypted hard disk. Another possibility is to use one-time passwords implemented with, e.g., the help of smart cards. Yet another possibility is to use hardware-based passwords on the client PCs.

Comment
Both spytstr and spy require the NetBIOS protocol. However, an installation of NetBIOS is a relatively simple operation made in less than five minutes by someone with prior Windows experience.

4.2.2 I2: Install Trojan Horse on Client to Snoop Keyboard Strokes

Synopsis
Keyboard snooping was accomplished by logging keyboard strokes with a TSR program. Such a program is activated for each keyboard interrupt.
Attack method
TSR programs can be used to record keystrokes in a log file. The log file may also be encrypted to hide its contents. *keytrap.com*, *log.com* and *keycopy.com* are all examples of publicly available tools for this purpose.

Cause
The insecure client configuration is again the culprit.

Attacker gain
An attacker can gain passwords and other secret information.

Remedy
Boot the clients via the network or boot from a write-protected or encrypted hard disk. Other possibilities are to use either one-time or hardware-based passwords on the client PCs.

4.2.3 I3: Install Trojan Horse on Client to Capture Passwords

Synopsis
It was impossible to capture plaintext passwords on the network because they were encrypted. However, it was possible to replace the NetWare login program on client PCs with a Trojan horse. A Trojan horse, in computer terms, is a program that performs a hidden function in addition to its stated function.

Attack method
First, the original login program was replaced by a Trojan horse. In this case, the hidden function was to record passwords, and the stated function was to offer a login service. Once the login name and the password were input to the Trojan horse, the original login program was started with the login name as argument. The password, on the other hand, had to be sent to the login program through the keyboard buffer. This was accomplished simply by writing back the password to the keyboard buffer.

Cause
The insecure client configuration causes the vulnerability.

Attacker gain
Anyone who gains physical access to a client PC can replace the NetWare login command with a Trojan horse that also records the login parameters.
Remedy
Boot the clients via the network or boot from a write-protected or encrypted hard disk. Other possibilities are to use either one-time or hardware-based passwords on the client PCs.

4.2.4 Connection Hijacking

Synopsis
A client’s communication with the file server can be hijacked, e.g., with the very well-known Dutch program called hack.exe.

Attack method
hack.exe exploits a deficiency in the NetWare Core Protocol (NCP). All system calls for network resources in NetWare are constructed as NCP packets, and one NCP packet holds information about which user has requested a service together with the user’s authentication ticket. The program extracts one NCP packet from the network and replaces the client network address with a well-selected address that can be arbitrarily selected. It then constructs 256 packets with different sequence numbers—sequence numbers are 1 byte long—and bombards the server with them, knowing that one packet will be accepted, thereby taking over the original user’s connection [18]. The program hack.exe exploits this deficiency by automatically taking over the supervisor’s communication.

Cause
The NetWare file server did not require the clients to authenticate their packets.

Attacker gain
An attacker can gain supervisor privileges.

Remedy
This flaw is, according to Novell, solved with NCP packet signatures, described in Section 3.5, where each NCP packet has an encrypted signature that changes with each packet.

Comment
It was necessary for the supervisor to be logged on to the system in order to hijack his connection.

4.3 Availability Attacks
If an attacker manages to force the system to deny service to authorized users, an availability attack has been performed. However, no such attacks were performed in the experi-
ment, which may be explained by the fact that availability attacks were, for obvious reasons, discouraged by the experimental supervisor team.

However, one such attack was suggested by a couple of groups. The attack was directed against the print server, which is an optional component installed on many NetWare file servers. Such a server offers a network print service to its connected clients. The suggested method was to overfill the print server partition with garbage, which will cause the file server to hang.

4.4 Summary of Attacks
Information on how to attack a NetWare system was in most cases found in Internet sources. It is notable that all groups succeeded in finding at least one way to break into the system.

The most successful attack method, all categories, was to attack the clients, i.e., to monitor users and install malicious software such as Trojan horses. The network was also vulnerable, since NetWare sends all data except passwords in plaintext.

It is important that the NetWare file server is physically secured. Otherwise it is possible to boot, for example, MS-DOS and directly modify arbitrary system files. There is also a backdoor in NetWare that allows anyone who can gain physical access to the server to become the supervisor of the system. It is worth noting that the file server was not attacked during the experiment. The reason for this is that, since the clients are as vulnerable as they turned out to be, attackers did not find it worthwhile to spend their time attacking the server.

5 Security Evolution
Since the intrusion experiment, Novell has released both NetWare 4 and NetWare 5, and, according to the vendor, these versions offer enhanced security. The new security features are discussed below. Newly found weaknesses are also covered and, finally, we summarize our impressions of the Novell NetWare operating system.

5.1 NetWare Directory Service (NDS)
The new security architecture is centered around the NetWare Directory Service (NDS), which was first introduced in NetWare 4. NDS is a distributed database\(^1\) that provides access to all network resources, including users and groups. This means that the bindery

\(^1\) NDS is an implementation of the standard X.500 directory service.
database, described in Section 3.4.1, is no longer needed. One advantage of this new scheme is that it allows a user to perform a single login to a group of NetWare servers.

The internal structure of NDS is fairly well known, see [16], and consists of four core files in NetWare 4. Five files are used in NetWare 5 and they have been given slightly different names. However, the two versions have a similar internal data structure. Ordinary users as well as non-users must not be able to directly access the NDS files, since they contain information that should be protected, e.g., hashed versions of the passwords, which may be used in a dictionary attack or a brute force attack, since the hash algorithm used in NetWare is known, see [9].

Novell has added additional security components to NDS in NetWare 5. The new features are based mainly on Novell’s International Cryptographic Infrastructure (NICI). One of the new services provided is Secure Authentication Services (SAS). SAS is entirely built on NICI and provides Secure Sockets Layer (SSL) support. Yet another new component is the Public Key Infrastructure Services (PKIS), which gives the ability to generate and manage digital certificates.

5.2 Auditing

A real-time auditing facility, implemented through the AUDITCON utility, was first introduced in NetWare 4. Two main types of events may be recorded: NDS actions and volume actions. A large set of different events of both types is offered. File actions, such as create file/directory, open file, and delete file/directory, are typically volume events that may be recorded, while change password, change ACL, and log in/out user are all examples of recordable NDS events.

The implemented auditing facility is protected with passwords. These are different from ordinary user passwords, and a separate one is used for each volume being audited. Note that the audit passwords are not recoverable, which means that they must be recorded carefully. Since NetWare 4.1, the auditing files are stored in a hidden directory. The current audit log file plus a total of maximum 15 backup log files may be stored. Neither of them is editable.

5.3 Security Flaws in NetWare 4 and 5

In this section, we will take a closer look at vulnerabilities discovered in NetWare 4 and 5. The discussion concentrates on a hacking toolbox referred to as the Pandora toolbox, developed by the Nomad Mobile Research Center (NMRC) in Arlington, Texas. Pandora was originally initiated as a project whose goal was to prove that it is indeed possible to recover passwords from NDS, even though the vendor has claimed the opposite. Addi-

\[1\] The PKIS uses X.509 version 3 certificates and is compatible with X.509 version 1 and 2 certificates.
tional flaws were discovered during the project, most of them related to the insecurity of the NCP and IPX protocols.

The Pandora toolbox consists of a collection of scripts that exploits weaknesses in Novell NetWare. The most recent version is v.4, released in July 1998. However, the analysis below is based on Pandora v.3, for the reasons that it has been tested, verified, and announced by Novell, see http://www.novell.com/products/nds/pandora.html.

### 5.3.1 Password Guessing
First of all, a set of dictionary attack programs is shipped in Pandora v3, including crypt, crypt2, intruder, extract, and manipul8. These traditional dictionary attacks are still useful to any attacker.

### 5.3.2 NCP Spoofing
An NCP exploit script called GameOver, used to spoof the server to give the user GUEST admin privileges, is also included. The idea behind this attack is to bypass packet signatures. The attack is accomplished as follows. On NetWare servers, there is a special user called the “file server”, which is an internal connection to the NetWare server and is bounded to an IPX/Socket number. The “file server” user has many privileges but uses only signature level 1. To perform the attack, the IPX/Socket number for this connection is needed. This information can be retrieved with Novell’s publicly available Onsite tool. An attacker can then spoof a bindery request that gives user GUEST admin rights.

There is also a program called Havoc, which causes denial of service. Again, the file server connection described above is the target. By sending illegal NCP calls to that connection, the NetWare server started broadcasting information on these illegal actions and, eventually, all connections were dropped.

### 5.3.3 Connection Hijacking
Two additional exploit scripts, Level1-1 and Level3-1, may also be used to gain administrator rights. However, Novell did not manage to get these two scripts to work as expected. Still, both are so-called connection hijacking attacks and, to successfully perform them, an administrator must be logged in to the system. According to [10], both attack methods are based on the fact that IPX fragmented request/replies are not signed.

### 5.4 Summary
Despite the fact that NetWare has security mechanisms of a rather high level, it was, and still is, quite easy for unskilled attackers to successfully penetrate such a system. Publicly available exploit scripts make technically advanced attacks for novice attackers possible.
The problems are caused mainly by insecure clients and weaknesses in the communication protocols, i.e., IPX and NCP.

Most attacks carried out in the intrusion experiment were directed against the insecure clients, which were configured with MS-DOS 6.2 and Windows 3.1. Today, such clients typically execute either Windows 95 or Windows 98, but this fact alone does not greatly strengthen the security of the system. Furthermore, weaknesses in IPX and NCP have successfully been exploited in a number of attacks. One example of a very serious weakness is the fact that it still is possible to hijack a connection to the NetWare server.

Since version 3.12, the vendor has made attempts to enhance the security of the system. First of all, a new audit facility was introduced in version 4. Secondly, the inclusion of the NetWare Directory Service (NDS) should also strengthen security, since it replaces the bindery database, which has been a target in many previous attacks. In spite of that, flaws in NDS have recently been discovered and announced. Finally, we note that the new cryptographic infrastructure integrated with NDS in NetWare 5 is promising from a security viewpoint.

6 Conclusion

It is clear that Novell Netware 3.12 exhibits a number of serious security flaws, most of which could be referred to the insecure clients and the communication network. It is also clear that more recent versions of the system present significant security improvements. Despite this, there are still many remaining vulnerabilities. One explanation for this may be that the vendor’s attempt to offer backward compatibility leaves open old weaknesses in the system. Another explanation is that most attacks are nowadays publicly announced and described in detail on the Internet. In many cases, ready-for-use exploit scripts are available and may be directly downloaded.

A staggering interpretation of this might be that security enhancements of widespread, commercially available systems will not be effective until they reach an extremely high level. The reason for this is that, even if the system is indeed improved, the total amount of effort spent to crack it is so high that some remaining weaknesses will always be found. And, since the information exchange within the hacker community is rapid and effective, the “benefits” of the accumulated intrusion effort spent are widely spread. For a discussion of effort spent for attacking purposes see [11, 17]. Thus, a disquieting future scenario would be that weaknesses will always remain in commercial systems and will sooner or later be discovered by attackers and exploited.
Acknowledgments

We are grateful to all the students that participated in the experiment in the Applied Computer Security course, 1994.
References


“Efterklokhet är en exakt vetenskap”

“Afterthought is an exact science”

— Hans Dahlborg1

1) English translation by the thesis author.