Intrusion Tolerance for Unclassified Networked Systems

Yves Deswarte & David Powell
LAAS-CNRS
7 avenue du Colonel Roche
31077 Toulouse cedex 4
France

{Yves.Deswarte, David.Powell}@laas.fr

ABSTRACT

Information such as security advisories, emergency recommendations, e-government information, etc., is unclassified, but its availability and integrity may be vital. Such data are intended to be made widely available and thus need to be accessible through open networks such as the Internet. The systems distributing this kind of information are usually built from COTS hardware and software, since their functions do not require specific software or hardware development. Openness and use of COTS make these systems very vulnerable, and traditional security means are insufficient to achieve the required availability and integrity. In that case, fault tolerance can be viewed as a complementary, valuable technique to cope with possible intrusions, as well as accidental failures of system components.

This paper presents the techniques of intrusion tolerance, and describe some recent experimental architectures, developed by the European project MAFTIA and the DARPA project DIT.

1 INTRODUCTION

Many systems that store, process or distribute unclassified but vital information, are connected to open networks, such as the Internet, in order to interact easily with other systems, or to give a wide public access to important information: security advisories, emergency recommendations, e-government information, etc.

Even though confidentiality is not critical for these systems since their information is unclassified, integrity and availability might be vital, in particular in emergency or crisis situations. On the other hand, most of these systems use COTS hardware and software, for economic reasons, but also because these systems support classic widely-deployed applications, which do not require specific software or hardware development.

These two characteristics, open network connection and COTS hardware and software, make these systems very vulnerable to attacks, while at the same time attacks are becoming ever more frequent on the Internet. Facing this evolution, the usual security techniques are insufficient and fault-tolerance techniques are increasingly worthwhile. Nevertheless, intrusions are different from accidental faults and specific fault-tolerance techniques must be designed to cope with their peculiarities.

In particular, the fault independence assumptions, which can be justified for the kind of accidental faults that are addressed by most fault tolerance techniques, are not valid for deliberate attacks. This means that if one kind of intrusion can be successful on a part of the system, the same kind of intrusion will be successful on other similar parts. Consequently, hardware and software diversification are essential.

Paper presented at the RTO IST Symposium on "Adaptive Defence in Unclassified Networks", held in Toulouse, France, 19 - 20 April 2004, and published in RTO-MP-IST-041.

The first section of this paper discusses the limits of conventional security techniques. Then the principles and techniques of intrusion tolerance are presented in Section 3. Sections 4 and 5 are dedicated to some recent experimental architectures, respectively developed by the European project MAFTIA and the DARPA project DIT.

2 SHORTCOMINGS OF CONVENTIONAL SECURITY TECHNIQUES

Computer and communication security relies mostly on user authentication and authorization, i.e. control of access rights. Authentication is necessary to identify each user with sufficient confidence, in order to assign him adequate privileges and to make him responsible and liable for his actions. Authorization aims to allow the user to perform only legitimate actions. As much as possible, authorization should obey the *least privilege principle*: at any time, a user can only perform the actions needed to achieve the task duly assigned to him. Authorization is implemented through protection mechanisms, which aim to detect and block any attempt by a user to exceed his privileges. Security officers can then detect such attempts and initiate legal actions, which in turn constitute deterrence against further attempts. Authentication, authorization, detection, retaliation and deterrence constitute the weapons of security defenders.

Unfortunately, these weapons are of little efficiency in the context considered in this paper:

- Unclassified information systems are often supposed to be accessed by a wide public, making strong authentication infeasible.
- COTS operating systems and application software contain many design flaws that can be exploited by attackers to circumvent protection mechanisms; when software companies develop and distribute patches to correct such flaws, relatively few system administrators apply the patches either because this would require more time or competence than available, or because the patches may disable certain features needed by other legitimate software.
- Most Internet protocols were designed thirty years ago, at a time when computing and communication resources were expensive and unreliable, and when intrusions were unlikely; so communication availability was the primary objective. Many facilities developed for that purpose can be diverted by malicious agents to perform denial of service attacks (e.g., by SYN flooding) or to multiply their efficiency (e.g., by smurfing), to by-pass protection mechanisms such as firewalls (e.g., by source routing), to hide their tracks (e.g., by IP address spoofing), etc.
- Due to harsh competition, most Internet Service Providers and telecommunication operators do not implement ingress filtering and trace-back facilities, which would help to locate and identify attackers.

3 INTRUSION TOLERANCE TECHNIQUES

3.1 Fault tolerance

Fault tolerance [1] is a technique that has proven to be efficient to implement computing systems able to provide a correct service despite accidental phenomena such as environment perturbations (external faults), failures of hardware components (internal physical faults), or even design faults such as software bugs.

According to the dependability terminology [2], *faults* are causes of errors, errors are abnormal parts of the computing system state, and *failures* happen when errors propagate through the system-to-user interface, i.e., when the service provided by the system is incorrect. When faults are accidental and sufficiently rare, they can be tolerated. To do so, errors must be *detected* before they lead to failure, and then corrected or *recovered*: this is the role of *error handling*. It is also necessary to diagnose the underlying faults (i.e., to identify and locate the faulty components), so as to be able to isolate them, and then replace or repair them, and finally to re-establish the system in its nominal configuration: fault diagnosis, isolation, repair and reconfiguration together constitute *fault handling*.

To detect errors, two main classes of techniques can be used. The first class is made of likelihood checks, which consist in observing the system state, in particular certain values or events, and verifying their likelihood. This usually imposes only a small hardware or software overhead (*redundancy*). Among hardware likelihood checks, let us note that most microprocessors detect non-existing or unauthorized instructions and commands, non-existing addresses and unauthorized access modes, and that watchdogs can detect excessive execution durations. Software likelihood tests can be inserted into programs to check the values of certain variables, or the instants or sequences of certain events (*defensive programming*). Some error detecting codes can also be viewed as likelihood checks.

The other main class of error detection techniques consists in comparing several executions, carried out either sequentially on the same hardware, or on different hardware units. This requires more redundancy than the first class of error detection techniques, but it also assumes that a single fault would not produce the same effect (i.e., identical errors) on the different executions. If only internal physical faults are considered, the same computation can be run on identical hardware units, since it is very unlikely that each hardware unit would suffer an identical internal fault at the same execution instant to produce the same error. On the contrary, design faults would produce the same errors if the same process is run on identical hardware units, and thus the comparison of the executions would not detect discrepancies. In that case, it is necessary to diversify the underlying execution support (hardware and/or software), so that a single design fault would affect only one execution, or at least would affect differently the different executions [3].

To correct errors, one approach it to take the system back to a state that it had occupied prior to the detection of errors, i.e., to carry out rollback recovery. To be able to do that, it is necessary to have created and saved copies of the system state, known as recovery points or *checkpoints*. Another error correction technique is called *forward recovery*, which consists of replacing the erroneous system state by a new, healthy state, and then continuing execution. This is possible, for example, in certain real-time control systems in which the system can be re-initialized and input data re-read from sensors before continuing execution. Finally, a third technique consists in “masking” errors; This is possible when there is enough redundant state information for a correct state to be built from the erroneous state, e.g., by a majority vote on three (or more) executions.

In most cases, the efficacy of fault tolerance techniques relies on the fact that faults are rare phenomena that occur at random points in time. It is thus possible, for example in a triple modular redundant architecture, to suppose that is unlikely for a second unit to fail while a failed unit is being repaired. This hypothesis is unfortunately not valid when intrusions are considered. An attacker that succeeds in penetrating one system can pursue his attack on that system, and also simultaneously attack other similar systems.

3.2 Vulnerability, attack, intrusion

An intrusion occurs when an attack is able to successfully exploit a vulnerability (a design or configuration fault, in the terminology of dependability) [2]. The intrusion may be considered as an internal fault, which can cause errors that may provoke a system security failure, i.e., a violation of the system’s security policy. As discussed earlier, it would be illusory to imagine that attacks over the Internet can be prevented. Similarly, it is impossible to eliminate all possible vulnerabilities. For example, the very fact that a system is connected to the Internet is in itself a vulnerability, but what use would a Web server be if it were not connected to the net?

It is therefore of interest to *tolerate intrusions*, i.e., to arrange things such that an intrusion in one part of the system has no consequence on its overall security. To do that, we can use techniques developed in the traditional field of fault tolerance. However, there are two main problems:

- It should be made very difficult for the same type of attack to succeed in different parts of the system. This means that each "part" of the system must be sufficiently protected in its own right (so that there are no trivial attacks), and should ideally be diversified.
- An intrusion into a part of the system should not allow the attacker to obtain confidential data. This is especially important in that redundancy, which is necessary for fault tolerance, may result in more alternative targets for hackers to attack.

If these problems can be solved, we can apply to intrusions the techniques that have been developed for traditional fault tolerance: error handling (detection and recovery) and fault handling (diagnosis, isolation, repair, reconfiguration). In the context of intrusions, specific detection techniques have been developed. These have been named "intrusion detection" techniques, but it should be noted that they do not directly detect intrusions, but only their effects, i.e., the errors due to intrusions (or even due to attacks which did not successfully cause intrusions).

The so-called intrusion detection techniques may be divided into two categories: anomaly detection and misuse detection (see Figure 1). Anomaly detection consists in comparing the observed activity (for example, of a given user) with a reference "normal activity" (for the considered user). Any deviation between the two activities raises an alert. Conversely, misuse detection consists in comparing the observed activity with a reference defining known attack scenarios. Both types of detection techniques are characterized by their proportions of false alarms (known as false positives) and of undetected intrusive activities (known as false negatives). In the case of anomaly detection, one can generally adjust the "threshold" or, by analogy with radar systems, the "gain" of the detector to choose a point of operation that offers the best compromise between the proportions of false positives and false negatives. On the other hand, misuse detection techniques have the advantage of identifying specific attacks, with few false positives. However, they only enable the detection of known attack symptoms. In both cases, it should be noted that detection is based on likelihood checks.

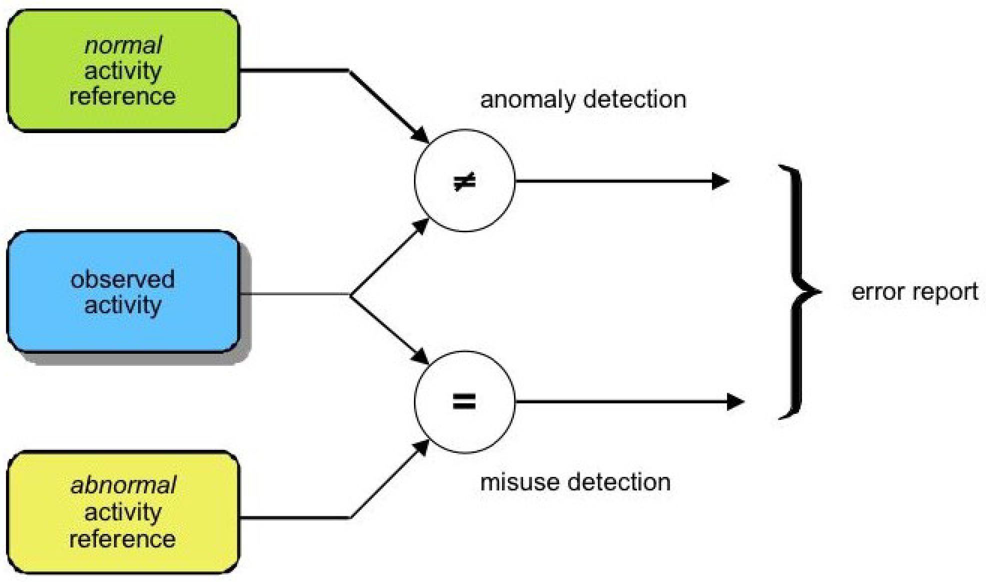

Figure 1: Intrusion detection paradigms

To correct the damage caused by the intrusion, one may, like in traditional fault tolerance, carry out backward recovery (if one has taken the precaution of maintaining up-to-date backups) or forward recovery (if one can rebuild a healthy state), but it is often easier and more efficient to mask errors, using some form of active (or modular) redundancy.

3.3 Fragmentation, redundancy, scattering

Several years ago, we developed an error masking technique, called "fragmentation, redundancy and scattering, or FRS", aimed at protecting sensitive data and computations [4]. This technique exploits distribution of a computing system to ensure that intrusion into part of the system cannot compromise the confidentiality, integrity and availability of the system. Fragmentation consists of splitting the sensitive data into fragments such that a single isolated fragment does not contain any significant information (confidentiality). The fragments are then replicated so that the modification or the destruction of fragment replicas does not impede the reconstruction of correct data (integrity and availability). Finally, scattering aims to ensure that an intrusion only gives access to isolated fragments. Scattering may be: *topological*, by using different data storage sites or by transmitting data over independent communication channels, or *temporal*, by transmitting fragments in a random order and possibly adding false padding fragments. Scattering can also be applied to privileges, by requiring the cooperation of several persons with different privileges in order to carry out some critical operation (separation of duty).

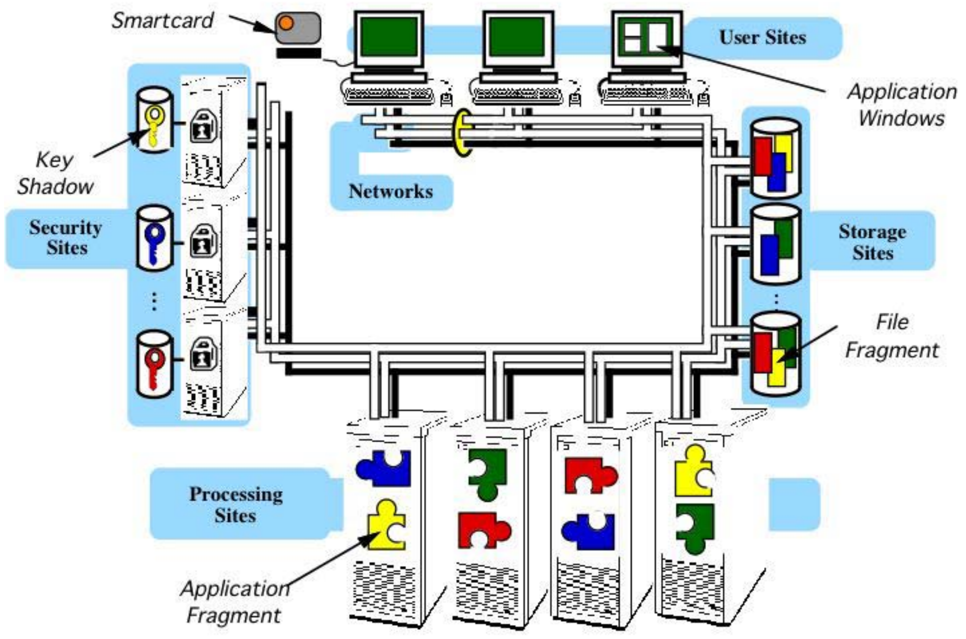

Figure 2 — Fragmentation-replication-scattering in Delta-4

The FRS technique was originally developed in the Delta-4 project [5] for file storage, security management and data processing (see Figure 2). For file storage, fragmentation is carried out using simple cryptographic techniques and fragment naming employs a secret key one-way function. The fragments are sent over the network in a random order, which means that one of the hardest tasks for an intruder would be to sort all the fragments into the right order before being able to carry out cryptanalysis. For security management, the principle resides in the distribution of the authentication and authorization functions between a set of sites administrated by different people so that failure of a few sites or misfeasance by a small number of administrators do not endanger the security functions. On these sites, non-sensitive data is replicated, whereas secret data is fragmented using threshold cryptographic functions. Finally, for data processing, two data types are considered: a) numerical and logical data, whose semantics are defined by the application, (b) contextual data (e.g., character strings) that is subjected only to simple operations (input, display, concatenation, etc.). In this scheme, contextual data is ciphered and deciphered only on a user site during input and display. In contrast, context data is subjected to successively finer fragmentation until the fragments do not contain any significant information. This is achieved using an object-oriented decomposition method.

3.4 The MAFTIA project

The techniques developed in Delta-4 are well adapted to predominately homogeneous applications that are distributed over a LAN. However, they are not directly transposable to Internet, especially when the concerned applications involve mutually suspicious companies or organizations. In this case, it is no longer possible to manage security in a homogeneous way.

The European project MAFTIA was directly aimed at the development of intrusion-tolerant Internet applications [6]. Protocols and middleware were developed to facilitate the management of fault-tolerant group communications (including tolerance of Byzantine faults), possibly with real-time, confidentiality and/or integrity constraints [7, 8, 9]. In particular, the developed protocols and middleware enabled the implementation of trusted third parties or TTPs (e.g., a certification authority) that tolerate intrusions (including administrator misfeasance) [10]. Particular attention was paid to intrusion detection techniques distributed over Internet, since intrusion detection not only contributes to intrusion tolerance, but is itself an attractive target for attack. It is thus necessary to organize the intrusion detection mechanisms in such a way as to make them intrusion-tolerant [11]. Furthermore, the project developed an authorization scheme for applications involving mutually suspicious organizations [12]. An authorization server, implemented as an intrusion-tolerant TTP, checks whether each multiparty transaction is authorized. If that is so, the server generates the authorization proofs that are necessary for the execution of each component of the transaction (invocations on elementary objects). On each of the sites participating in the authorization scheme, a reference monitor, implemented on a JavaCard, checks that each method invocation is accompanied by a valid authorization proof. The scheme is intrusion-tolerant in the sense that the corruption of a participating site does not allow the intruder to obtain any additional privileges regarding objects residing on other sites [12].

3.5 The DIT project

In cooperation with SRI International, we are participating in the development of the DIT (Dependable Intrusion Tolerance) architecture [13]. The objective is to be able to build Web servers that continue to provide correct service in the presence of attacks. For this type of application, confidentiality is not essential, but integrity and availability must be ensured, even if the system is under attack from competent attackers. It is thus essential that a successful attack on one component of the system should not facilitate attacks on other components. The architecture design is thus centered on a diversification approach.

The architecture is composed of a pool of ordinary Web servers, using as much diversification as possible at the hardware level (Sparc, Pentium, PowerPC, etc.), the operating system level (Solaris, Microsoft Windows, Linux, MacOS, etc.) and Web application software level (Apache, IIS, Enterprise Server, Openview Server, etc.) (see Figure 3). Only the content of the Web pages is identical on each server. There are sufficient application servers at a given redundancy level (see below) to ensure an adequate response time for the nominal request rate. The servers are isolated from the Internet by *proxies*, which are implemented by purpose-built software executed on diversified hardware. Requests from the Internet, filtered by a firewall, are taken into account by one of the proxies acting as a *leader*. The leader distributes the requests to multiple Web servers and checks the corresponding responses before returning them to the request initiator. The back-up proxies monitor the behavior of the leader by observing the firewall/proxy and proxy/server networks. If they detect a failure of the leader, they elect a new leader from among themselves. The proxies also process alarms from intrusion detection sensors placed on the Web servers and on both networks.

Depending on the current level of alert, the leader sends each request to one server (simplex mode), two servers (duplex mode), three servers (triplex mode) or to all available servers. Each server prepares its response and then computes an MD5 cryptographic checksum of this response and send it to the leader. In simplex mode, the server also sends its response to the leader, which recomputes the checksum and compares it to the one sent by the server. In duplex mode, the leader compares the two checksums from the servers and, if they concur, requests one the responses, which is verified by recomputing the checksum. In triplex or all-available modes, the checksums are subjected to a majority vote, and the response is requested from of the majority servers.

The alert level is defined as either a function of recent alarms triggered by the intrusion detection mechanisms or other error detection mechanisms (result cross-checking, integrity tests, etc.), or by information sent by external sources (CERTs, other trusted centers, etc). The redundancy level is raised towards a more severe mode (higher redundancy level) as soon as alarms are received, but reverts to a less severe mode (lower redundancy level) when failed components have been diagnosed and repaired, and when the alarm rate has decreased. This adaptation of the redundancy level is thus tightly related to the detection, diagnosis, reconfiguration and repair mechanisms. In the case of read-only data servers, such as passive Web servers, repair involves just a simple re-initialization of the server from a back-up (an authenticated copy on read-only storage).

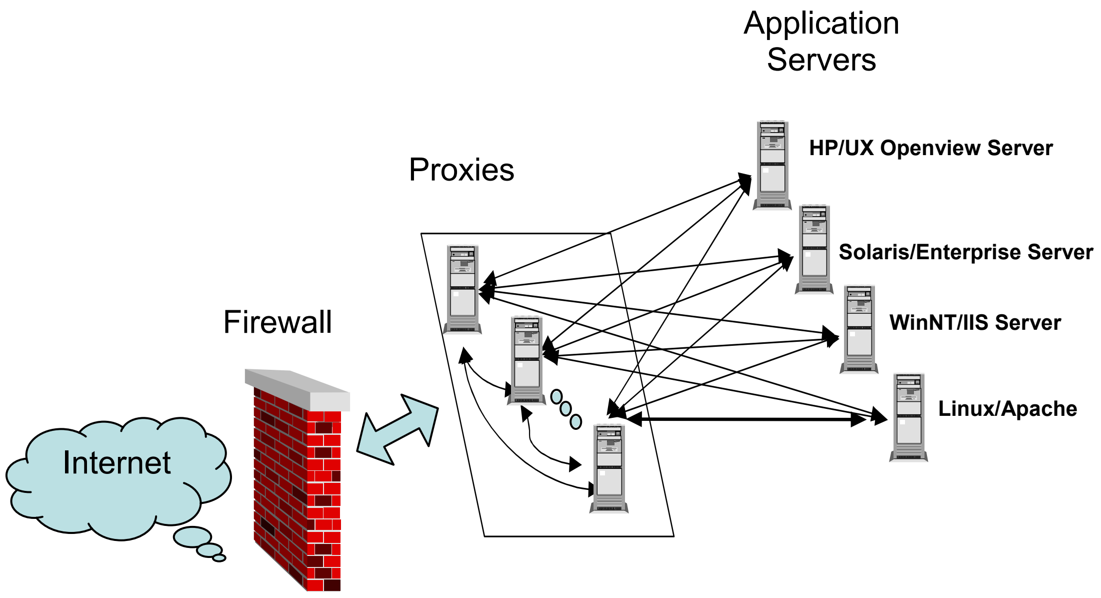

Figure 3: DIT architecture

Diversification renders the task of the attacker as difficult as possible: when an attacker sends a Web page request (the only means for him to access the application servers), he does not know towards which servers his request will be forwarded and thus which hardware or software will process it. Even if he were able to design an attack that would be effective on all server types (except maybe for denial-of-service attacks, which are easy to detect), it would be very difficult to cause redundant servers (in duplex mode and above) to reply in exactly the same incorrect way.

4 CONCLUSION

Given the current rate of attacks on Internet, and the large number of vulnerabilities in contemporary computing systems, intrusion tolerance appears to be a promising technique for implemented more secure applications, particularly with diversified hardware and software platforms. There is of course a price to pay, since it is expensive to support multiple heterogeneous systems. However, this is probably the price that must be paid for security in an open, and therefore, uncertain world.

5 BIBLIOGRAPHY

[1] J. Arlat, Y. Crouzet, Y. Deswarte, J.-C. Laprie, D. Powell, P. David, J.-L. Dega, C. Rabéjac, H. Schindler, J.-F. Roucailles, "Fault Tolerant Computing", in *Encyclopedia of Electrical and Electronic Engineering*, John G. Webster, Ed., Wiley-Interscience, Volume 7, 1999, pp.285-313.

[2] A. Avizienis, J.-C. Laprie, B. Randell, "Fundamental Concepts of Dependability", in *Proc. 3rd Information Survivability Workshop*, IEEE CS Press, 24-26 October 2000, Boston, MA, USA, pp. 7-12.

[3] Y. Deswarte, K. Kanoun, J.-C. Laprie, "Diversity against accidental and deliberate faults", in *Computer Security, Dependability and Assurance*: *From needs to Solutions*, P. Amman, B.H. Barnes, S. Jajodia & E.H. Sibley Eds., IEEE Computer Society Press, 1999, pp. 171-182.

[4] Yves Deswarte, Laurent Blain, Jean-Charles Fabre, "Intrusion Tolerance in Distributed Systems", *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Oakland (USA), 20-22 May 1991, pp. 110-121.

[5] D. Powell, Ed., *Delta-4: a Generic Architecture for Dependable Distributed Computing,* in Research Reports ESPRIT series, Springer-Verlag, 1991, ISBN 3-540-54985-4, 484 pages.

[6] D. Powell, A. Adelsbasch, C. Cachin, S. Creese, M. Dacier, Y. Deswarte, T. McCutcheon, N. Neves, B. Pfitzmann, B. Randell, R. Stroud, P. Veríssimo, M. Waidner, "MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications)", *Sup. of the Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN2001)*, Göteborg (Sweden), 1-4 July 2001, pp. D-32-D-35.

[7] Michael Backes and Christian Cachin, "Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries", in *Proc. Intl. Conference on Dependable Systems and Networks (DSN-2003)*, IEEE Computer Society Press, Washington, CA, USA, June 2003, pp. 37-46.

[8] M. Correia and L. C. Lung and N. F. Neves, P. Veríssimo, "Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model", in *Proceedings of the 21th IEEE Symposium on Reliable Distributed Systems (SRDS 21)*, IEEE Computer Society Press, October 2002, Suita, Japan, pp. 2-11.

[9] M. Correia, P. Veríssimo, Nuno F. Neves, "The Design of a COTS Real-Time Distributed Security Kernel", in *Proceedings of the Fourth European Dependable Computing Conference* (*EDCC 4*), October 2002, Toulouse, France, pp. 234-252.

[10] Christian Cachin, "Distributing trust on the Internet", in *Proc. Intl. Conference on Dependable Systems and Networks (DSN-2001)*, IEEE Computer Society Press, Göteborg (Sweden), 1-4 July 2001, pp. 183-192.

[11] H. Debar and A. Wespi, “Aggregation and Correlation of Intrusion-Detection Alerts”, in *Proceedings of Recent Advances in Intrusion Detection (RAID 2001)*, LNCS 2212, Springer, 2001, W. Lee, L. Mé, A. Wespi (eds.), pp. 85-103.

[12] Yves Deswarte, Noreddine Abghour, Vincent Nicomette, David Powell, “An Intrusion-Tolerant Authorization Scheme for Internet Applications”, in *Sup. of the Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN2002)*, Washington, D.C. (USA), 23-26 June 2002, pp. C-1.1 — C-1.6.

[13] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stavridou, T. Uribe, “An Adaptative Intrusion-Tolerant Server Architecture”, *Proceedings of the 10th International Workshop on Security Protocols*, Cambridge (UK), April 2002, to appear in Springer LNCS Series.