Internet Security: An Intrusion-Tolerance Approach

YVES DESWARTE, MEMBER, IEEE, AND DAVID POWELL, MEMBER, IEEE

Invited Paper

The Internet has become essential to most enterprises and many private individuals. However, both the network and computer systems connected to it are still too vulnerable and attacks are becoming evermore frequent. To face this situation, traditional security techniques are insufficient and fault-tolerance techniques are becoming increasingly cost-effective. Nevertheless, intrusions are very special faults, and this has to be taken into account when selecting the fault-tolerance techniques.

Keywords*—*Computer security, dependability, fault tolerance, Internet, intrusion tolerance.

I. INTRODUCTION

Arpanet, the experimental network that was to give rise to the Internet, was created by a small group of researchers and computer scientists who wanted to improve communication between themselves and to share some rare and expensive resources such as processors, mass storage, and intercomputer communication lines. The protocols and services that they developed were aimed primarily at achieving the best possible availability, despite the use of relatively unreliable components (communication lines, routers, computers). They did not envisage any malicious use of the network, since it was only accessible to a small group of pioneers who had a common aim: making the network work. None of them would have had the crazy idea of experimenting with attacks endangering this common aim, since that would probably have led to them being immediately expelled from the group. It is thus natural that the developed protocols and services only took into account transmission errors and accidental faults affecting hardware and software, without worrying about malicious attacks and intrusions. For example, in those protocols, nothing guarantees the authenticity of addresses, which facilitates spoofing attacks and session hijacking. Furthermore, the protocols include network maintenance functions, such as source routing (a technique whereby the sender of a packet can specify the route that a packet should take through the network), which can easily be exploited to bypass protection mechanisms such as firewalls.

Today, the same protocols are used in the Internet, which has become a totally open system, i.e., one to which anybody can have access. The current users of the Internet are very different to those forming the small closely-knit community of the original Arpanet. There are interactions between many different user-categories: business-to-business (B2B), business-to-consumer (B2C), citizen-to-administration (C2A) or government (e-Government), or simply among private citizens who create their own virtual communities. The Internet is used for commercial, administrative, democratic, social, and cultural reasons, or simply for recreation. Most uses of the Internet are perfectly legitimate. No single group has the right to exclude another group under the pretext that the latter could prevent the former from achieving its objectives. Indeed, there would be no interest in doing so, since it is the very diversity of uses to which the Internet is put that enables it to exist at a reasonable cost. Since the objectives are different, it is quite normal that the security requirements are also different, and it would be unreasonable to expect a schoolchild to manage his personal computer with as much care and attention to security as would the administrator of a bank server, for example. There are many more schoolchildren than banks, so it is not surprising that many systems connected to the Internet are not well administered. Such machines are very vulnerable to attack from malicious persons or agents, who might attempt to take control of them to further propagate their attacks. It is thus possible for attackers not only to increase their firing power in order to perpetrate attacks targeted at well-protected sites, but also to hide their tracks and make it more difficult to find clues enabling them to be identified.

Attacks are very common, which is not really surprising: the users of the Internet are more or less representative of

Manuscript received October 1, 2004; revised January 3, 2005.

The authors are with LAAS-CNRS, Toulouse Cedex 4 31077, France (e-mail: Yves.Deswarte@laas.fr; David.Powell@laas.fr).

Digital Object Identifier 10.1109/JPROC.2005.862320

the populations of developed countries, and among the hundreds of millions of Internet users, there must be a nonnegligible proportion of individuals who are potential attackers. The most frequent types of attacks, and also the easiest ones, are those aimed against availability, by "denial of service": the attacker aims simply to prevent the targeted system from being used. Other attacks are aimed against confidentiality: the attacker aims to obtain sensitive information such as commercial, industrial, political, or even military secrets, but also personal data whose disclosure may endanger people's privacy. Yet other types of attacks are aimed against the integrity of information: destruction or modification of sensitive data, spreading of false information, manipulation of published data, etc. Thus, one of the "sports" currently popular among hackers is to "deface" Web servers, i.e., to alter (or deviate the access route to) legitimate Web pages so as to replace the displayed information by humoristic, polemic, or pornographic parodies.

Attackers have various motivations. They may act out of sport or by curiosity (to carry out experiments), by vanity (to show off their competence), by vandalism (for the pleasure of destruction or damage), by vengeance (to hurt people they do not like or to punish those who do not consider them for their "merit"), by greed (blackmail, extortion), or even for political, strategic, or terrorist reasons. Attackers thus vary in degrees of tenacity and competency, and are able to deploy different levels of resources, according to whether they are disturbed adolescents, more-or-less structured groups of hackers, simple thieves, criminal or terrorist organizations, or government services specialized in electronic warfare.

There are also many ways to carry out attacks. They may exploit vulnerabilities in networks or their protocols: eavesdropping (or "sniffing"), interception (message destruction, insertion, modification or replay), address falsification, injection of counterfeit network control messages (e.g., for routing), and denial of service (e.g., by network jamming). Routers are also becoming ever more frequent targets. Attacks can also exploit flaws in operating systems and application software, such as buffer or stack overflows. It should also be noted that the Internet is also a medium for spreading information about security vulnerabilities, both for hackers and for system administrators and developers. The former use this information to develop and publish new "exploits" (i.e., methods for carrying out successful attacks) or even scripts to carry them out automatically. The latter use the same information to develop and publish remedies and patches. Like Aesop's tongue, the distribution of such security information over the Internet can be both the best and the worst of things.

II. 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 given instant, a user can only perform the actions needed to carry out 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 arsenal of security defenders.

Unfortunately, these weapons are of little efficiency in the context of the Internet.

1) Any Internet user, even anonymous users, has some rights on connected machines: e.g., the capacity to know their existence and to identify them by their name or address, the ability to read pages on public Web servers, etc.
2) Many systems connected to the Internet are accessible by the public at large, making strong authentication infeasible. Weak mechanisms such as password authentication are often used carelessly, e.g., password lending, which makes it possible for one user to masquerade as another. Even without the cooperation of a user, it is often easy to guess passwords.
3) Commercial off-the-shelf (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.
4) Most Internet protocols were designed at a time when computing and communication resources were expensive and unreliable, and when intrusions were unlikely. As mentioned in the introduction, in those days (30 years ago), communication availability was the primary objective. Many of these legacy protocols 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 bypass protection mechanisms such as firewalls (e.g., by source routing), to hide their tracks (e.g., by IP address spoofing), etc.
5) Due to harsh economic pressures, many Internet service providers and telecommunication operators do not implement ingress filtering and trace-back facilities, which would help to locate and identify attackers.

III. A Different Perspective on Security Methods

Twenty years ago, a few pioneering researchers contended that a "tolerance" approach to security could be defined using concepts borrowed from the area of dependable computing and fault-tolerance. Notable work of this period includes that done by Laprie *et al.* [1], Deswarte *et al.* [2], Dobson and Randell [3], and Joseph and Avizienis

[4]. More recently, institutionally financed research efforts such as the DARPA Organically Assured and Survivable Information System (OASIS) program[1] and the European IST project on Malicious and Accidental Fault Tolerance for Internet Applications (MAFTIA)[2] have given new credence to the basic tenets of this early work.

Here, we briefly introduce and discuss two particularly relevant key ideas from the field of dependable computing and fault tolerance [5]–[7].

The first key concept is that three causally related impairments to system dependability need to be considered: fault, error, and failure. A system *failure* is an event that occurs when the service delivered by the system deviates from correct service. An *error* is that part of the system state that may cause a subsequent failure, whereas a *fault* is the adjudged or hypothesized cause of an error. The notion is recursive in that a failure at one level can be viewed as a fault at the next higher level (e.g., a component failure is a fault seen from the containing system).

The labels given to these concepts conform to standard usage within the dependability community, but the important point we would like to stress is not the words but the fact that there are *three* concepts.

First, it is essential to be able to distinguish the internally observable phenomenon (error) from the externally observable one (failure), which tolerance techniques aim to avert. Indeed, any tolerance technique must be based on some form of detection and recovery acting on internal perturbations before they reach the system's interface to the outside world. The alternative viewpoint, in which any detectable anomaly is deemed to make the system "insecure" in some sense, would make intrusion *tolerance* an unattainable objective.

Second, the distinction between the internally observable phenomenon (error) and its root cause (fault) is vital, since it emphasizes the fact that there may be various plausible causes for the same observed anomaly, including an atypical usage profile, an accidental fault or an intentionally malicious fault.

The type of intentionally malicious fault of particular concern here is an *intrusion*, which is a deliberate software domain[3] operational[4] fault. An intrusion occurs when an *attack* is able to successfully exploit a *vulnerability*. Attacks may be viewed either at the level of human activity (of the attacker), or at that of the resulting technical activity that is observable within the considered computer system. Attacks (in the technical sense) are malicious faults that attempt to exploit one or more vulnerabilities (e.g., e-mail viruses, malicious Java applets or ActiveX controls). The intrusion resulting from the successful exploitation of a vulnerability by an attack 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.

[1] [Online]. Available: http://www.tolerantsystems.org/

[2] [Online]. Available: http://www.research.ec.org/maftia/

[3]As opposed to faults in the hardware domain, e.g., physical sabotage.

[4]As opposed to faults introduced during system development, e.g., trapdoors.

The second key concept from dependable computing is that there is a set of complementary methods for designing and validating dependable systems, which can be broadly classified into four broad categories:

- *fault prevention*: how to prevent the occurrence or introduction of faults.
- *fault tolerance*: how to deliver correct service in the presence of faults.
- *fault removal*: how to reduce the number or severity of faults.
- *fault forecasting*: how to estimate the present number, the future incidence, and the likely consequences of faults.

Fault prevention and fault removal are sometimes grouped together as *fault avoidance*; fault tolerance and fault forecasting constitute *fault acceptance*. Note that avoidance and acceptance should be considered as complementary rather than alternative strategies.

It is enlightening to equate "fault" in these definitions with the notions of attack, vulnerability, and intrusion defined earlier. Taking "attack" in both its human and technical senses leads to ten distinct security-building methods out of a total of 16 (see Table 1), as explained below.

Equating *attack*, *vulnerability*, and *intrusion* with fault, we obtain three clearly distinguishable sets of fault-**prevention** methods:

- *Attack prevention* (human sense): how to prevent the occurrence of human attacks. With attack taken in the human sense, this includes deterrence measures such as social pressure, laws, and their enforcement.
- *Attack prevention* (technical sense): how to prevent the occurrence of technical attacks. When attack is taken in its technical sense, attack prevention consists of the introduction of mechanisms such as authentication, authorization, and firewalls, which prevent attacks in that they "push back" the attacks to the level of the additional barriers these mechanisms introduce.
- *Vulnerability prevention*: how to prevent the occurrence or introduction of vulnerabilities. This includes measures going from semiformal and formal specification, rigorous design, and system management procedures, up to and including user education (e.g., choice of passwords).

Note that *intrusion prevention* (as opposed to intrusion *tolerance*) can be seen as the combined application of attack and vulnerability prevention, as well as attack and vulnerability removal, i.e., the classic security techniques.

Fault **removal** may occur either during development (verification and validation) or after a system is put into operation (maintenance). Equating *attack*, *vulnerability*, and *intrusion* with fault leads to the following interpretations of fault removal.

- *Attack removal* (human sense): how to reduce the number or severity of human attacks. This covers human countermeasure techniques aimed directly against the attacker.
- *Attack removal* (technical sense): how to reduce the number or severity of technical attacks. This covers

Table 1
Classification of Security Methods

Method Category		Attack (human sense)	Attack (technical sense)	Vulnerability	Intrusion
Fault Avoidance	**Prevention** (how to prevent occurrence or introduction of…)	deterrence, laws, social pressure, secret service…	firewalls, authentication, authorization…	semi-formal & formal specification, rigorous design & management…	= attack & vulnerability prevention & removal
	Removal (how to reduce number or severity of…)	physical counter-measures, capture of attacker	preventive & corrective maintenance aimed at removal of attack agents	1. formal proof, model-checking, inspection, test… 2. preventive & corrective maintenance, including security patches	⊆ attack & vulnerability removal, i.e., preventive & corrective maintenance
Fault Acceptance	**Tolerance** (how to deliver correct service in the presence of…)	= vulnerability prevention & removal, intrusion tolerance		= attack prevention & removal, intrusion tolerance	error detection & recovery, fault masking, intrusion detection & response, fault handling
	Forecasting (how to estimate present number, future incidence, likely consequences of…)	intelligence gathering, threat assessment…	assessment of presence of latent attack agents, potential consequences of their activation	assessment of: presence of vulnerabilities, exploitation difficulty, potential consequences…	= vulnerability & attack forecasting

maintenance actions aimed at removing malicious logic acting, or capable of acting, as attack agents (i.e., some forms of malicious logic).

- *Vulnerability removal*: how to reduce the number or severity of vulnerabilities. During system development, this covers verification procedures such as formal proof, model checking, and testing, specifically aimed at identifying flaws that could be exploited by an attacker. Identified flaws may then be removed by correcting the code. During system operation, vulnerability removal corresponds to preventive and corrective maintenance actions such as applying a security patch, withdrawing a given service, changing a password, removal of malicious logic implementing a trapdoor, etc.

We find no meaningful separable interpretation of fault removal in terms of intrusions other than preventive and corrective maintenance procedures aimed at removing attack agents and vulnerabilities resulting from intrusions.

With respect to fault **tolerance**, equating *attack*, *vulnerability* and *intrusion* with fault does not lead to clearly distinguishable sets of methods. First, since an intrusion cannot occur in the absence of vulnerability, intrusion tolerance and vulnerability tolerance are equivalent in the sense that tolerance of an intrusion implies tolerance of the vulnerability or vulnerabilities that were exploited to perpetrate the intrusion. To conform to current usage, we will refer to *intrusion tolerance*. Note that the presence of vulnerabilities can be tolerated if no attacks occur, so attack prevention and removal are also a form of vulnerability tolerance. Similarly, attack tolerance does not define a separate set of methods beyond vulnerability prevention and removal, and intrusion tolerance. Hence, we obtain one distinguishable set of fault-tolerance methods:

- *Intrusion tolerance*: how to provide correct service in the presence of intrusions. Admitting that attack, vulnerability, and intrusion prevention measures are always imperfect, intrusion tolerance aims to ensure that the considered system provides security guarantees in spite of partially successful attacks.

Fault **forecasting** refers to methods aimed at ranking or evaluating the effectiveness of fault prevention, removal, and tolerance techniques Equating *attack*, *vulnerability*, and *intrusion* with fault, we obtain three clearly distinguishable sets of methods.

- *Attack forecasting* (human sense): how to estimate the present number, the future incidence and the likely consequences of (human) attacks. This includes intelligence gathering, threat assessment, and attack warning.
- *Attack forecasting* (technical sense): how to estimate the present number, the future incidence and the likely consequences of (technical) attacks. This corresponds to an assessment of the present number of latent attack agents, and the future incidence and the likely consequences of their activation.
- *Vulnerability forecasting*: how to estimate the present number, the future incidence, and the likely consequences of vulnerabilities. This includes the gathering of statistics about the current state of knowledge regarding system flaws, and the difficulties that an attacker would have to overcome in order to take advantage of them.

Security risk analysis can be viewed as a combination of all three forecasting methods.

As discussed earlier, it would be illusory to imagine that all attacks over the Internet might 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? The focus in this paper is therefore on intrusion-*tolerance* techniques, which should be seen as an additional defense mechanism rather than an alternative to the classic set of techniques grouped under the heading intrusion-*prevention* on Table 1. Before addressing intrusion tolerance *per se*, we first present some background material on fault tolerance in the traditional sense of dependable computing.

IV. Fault Tolerance

Fault tolerance [8] is a technique that has proven to be efficient to implement computing systems able to provide a correct service despite accidental phenomena such as environmental perturbations (external faults), failures of hardware components (internal physical faults), or even design faults such as software bugs.

As outlined in Section III, *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 reestablish the system in its nominal configuration: fault diagnosis, isolation, repair, and reconfiguration together constitute *fault handling*.

There are various techniques for detecting errors. For simplicity, we categorize these as being either property checks or comparison checks.

Property checks consist in observing the system state, in particular certain values or events, and verifying they satisfy certain properties or rules. This usually imposes only a small hardware or software overhead (*redundancy*). Among hardware property checks, let us note that most microprocessors detect nonexisting or unauthorized instructions and commands, nonexisting addresses and unauthorized access modes, and that watchdogs can detect excessive execution durations. Software-based property checks include likelihood tests inserted into programs to check the values of certain variables, or the instants or sequences of certain events (*defensive programming*). Error detecting codes and run-time model checking can also be viewed as property checks.

Comparison checks consist 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 [9]–[11].

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 reinitialized and input data reread 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.

V. Intrusion Tolerance

Intrusion tolerance aims to organize and manage a system 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.

First, 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.

Second, 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

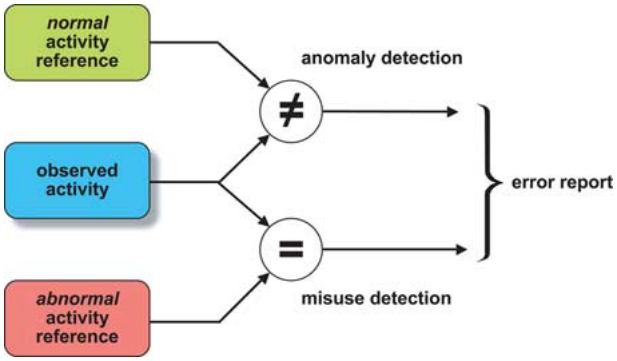

Fig. 1. Intrusion detection paradigms.

tolerance: error handling (detection and recovery) and fault handling (diagnosis, isolation, repair, reconfiguration).

A. *Tolerance Based on Intrusion Detection*

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 Fig. 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 property checks.

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.

B. *Fragmentation, Redundancy, and Scattering*

Several years ago, we developed an error masking technique, called fragmentation, redundancy, and scattering (FRS), aimed at protecting sensitive data and computations [12]. 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).

The FRS technique was originally developed in the Delta-4 project [13] for file storage, security management and data processing (see Fig. 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 administered 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, nonsensitive data is replicated, whereas secret data is fragmented using threshold cryptographic functions. Finally, for data processing, two data types are considered: 1) numerical and logical data, whose semantics are defined by the application, or 2) 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.

VI. The Internet Context

The techniques developed in Delta-4 are well adapted to predominately homogeneous applications that are distributed over an LAN. However, they are not directly transposable to the 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. Here, we briefly outline two projects were the tolerance approach has been adapted to account for the inherent heterogeneity of the Internet.

A. *The MAFTIA Project*

The European project MAFTIA was directly aimed at the development of intrusion-tolerant Internet applications [14]. Protocols and middleware were developed to

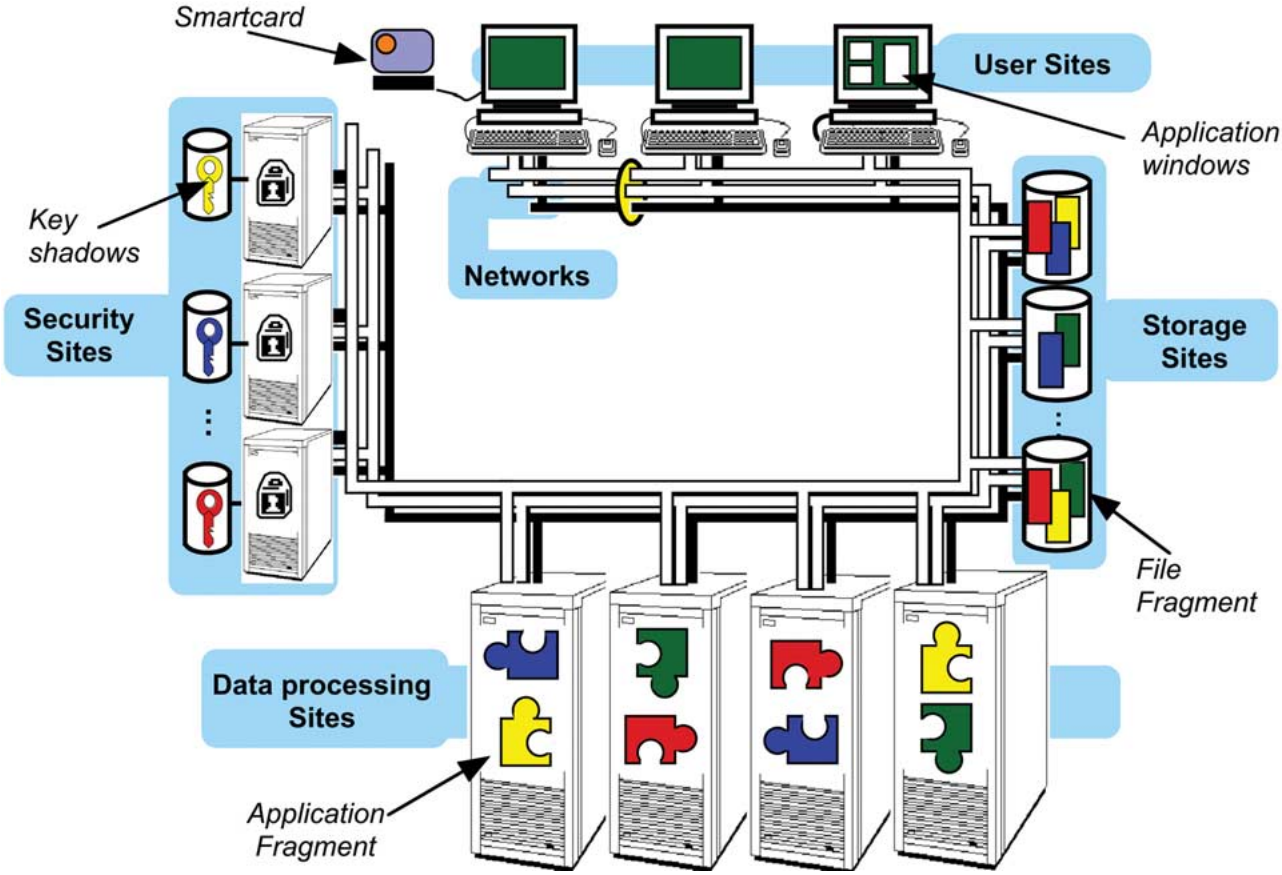

Fig. 2. FRS in Delta-4.

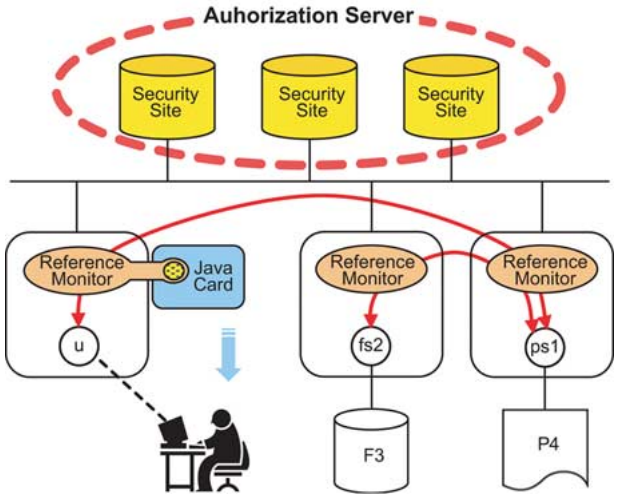

Fig. 3. MAFTIA authorization scheme.

facilitate the management of fault-tolerant group communications (including tolerance of Byzantine faults), possibly with real-time, confidentiality, and/or integrity constraints [15]–[18]. In particular, the developed protocols and middleware enabled the implementation of trusted third parties (TTPs) (e.g., a certification authority) that tolerate intrusions (including administrator misfeasance) [19] through error masking. Particular attention was also 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 [20]. Furthermore, the project developed an authorization scheme for multiparty transactions involving mutually suspicious organizations (see Fig. 3) [21].

An authorization server, implemented as an intrusion-tolerant TTP, checks whether each multiparty transaction is au-

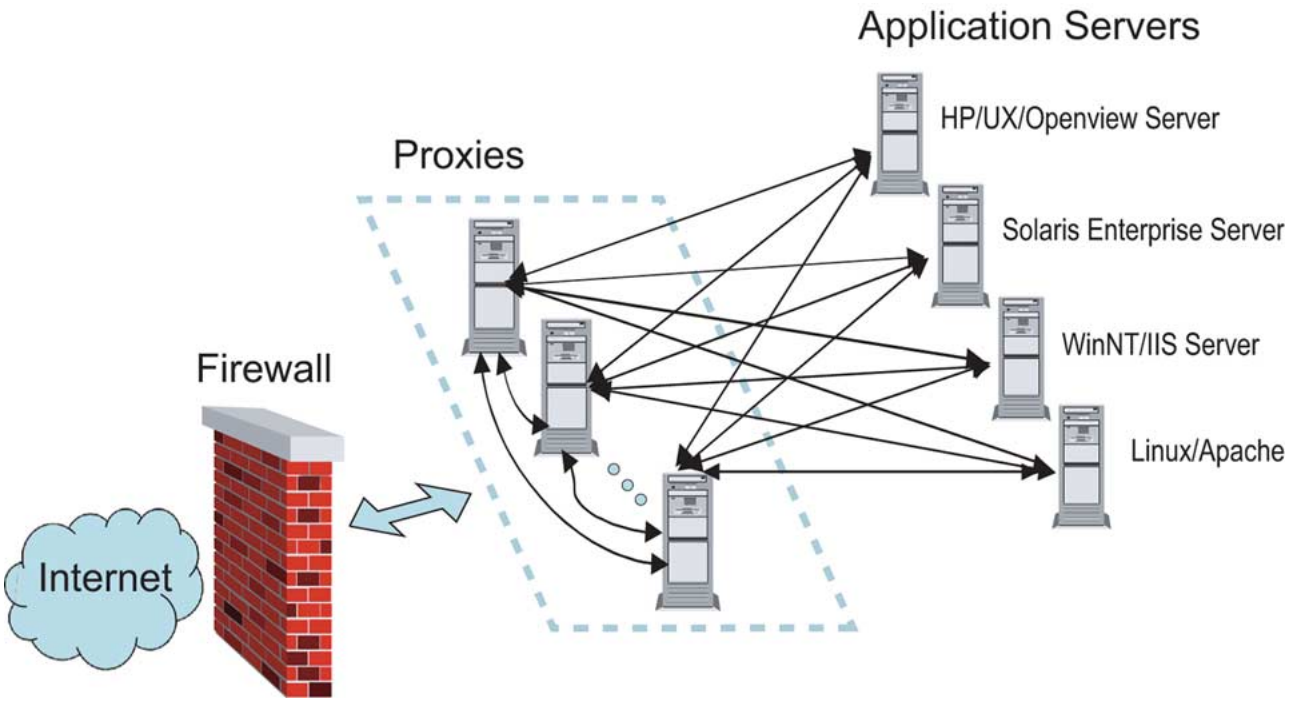

Fig. 4. DIT architecture.

thorized. 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.

B. The DIT Project

In cooperation with SRI International, we are participating in the development of the Dependable Intrusion Tolerance (DIT) architecture [22]. 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 (Fig. 4).

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.). 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 backup 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 all available servers. Each server prepares its response, then computes an MD5 cryptographic checksum of this response and sends 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 one 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 toward a more severe mode as soon as alarms are received, but is lowered to a less severe mode 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 reinitialization of the server from a backup (an authenticated copy on read-only storage).

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 toward 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.

VII. Related Work

The idea of applying fault-tolerance to security problems was developed in the 1980s [3], [23]. The first intrusion-tolerance techniques were developed for storing files, either by FRS [24] or by *information dispersal* [25]. Later, intrusion-tolerance techniques were also applied to security servers (authentication and authorization) [26] and data processing [27]. More recently, threshold crypto has been applied to intrusion-tolerant file storage [28] or to the storage and distribution of user secrets [29].

The recent development of intrusion-tolerant architectures is related to the growth of Internet insecurity. In Europe, this has been the motivation for the MAFTIA project [14] of the European IST research program, while in the United States, DARPA created the OASIS program [30], which allowed exploration of many intrusion-tolerance techniques, from wrappers [31] to proof-carrying code, and the establishment of new survivability paradigms, such as unpredictable adaptation [32]. The main applications of intrusion tolerance have been to TTPs [19], certification authorities [33], Web servers [22] and databases [34], with most of these architectures using COTS components [35].

VIII. 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 to implement 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.

References

[1] J.-C. Laprie, "Dependable computing and fault tolerance: concepts and terminology," in *Proc. 15th IEEE Int. Symp. Fault Tolerant Computing (FTCS-15)* 1985, pp. 2–11.

[2] Y. Deswarte, J.-C. Fabre, J. da Silva Fraga, J.-C. Laprie, and D. Powell, "The SATURNE project. A fault- and intrusion-tolerant distributed system," *IEEE Comput. Arch. Tech. Comm. Newslett.,* pp. 4–22, 1985.

[3] J. E. Dobson and B. Randell, "Building reliable secure systems out of unreliable insecure components," in *IEEE Symp. Security and Privacy* Oakland, CA, USA, 1986, pp. 187–193.

[4] M. K. Joseph and A. Avizienis, "A fault tolerance approach to computer viruses," in *Proc. IEEE Symp. Security and Privacy* 1988, pp. 52–58.

[5] J.-C. Laprie, "Dependability: Basic concepts and terminology," in *Dependable Computing and Fault-Tolerance*, A. Avizienis, H. Kopetz, and J.-C. Laprie, Eds. Vienna, Austria: Springer-Verlag, 1992, vol. 5, p. 265.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Secure Comput.,* vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[7] A. Avizienis, J.-C. Laprie, and B. Randell, "Fundamental concepts of dependability," in *Proc. 3rd Information Survivability Workshop* 2000, pp. 7–12.

[8] J. Arlat, Y. Crouzet, Y. Deswarte, J.-C. Laprie, D. Powell, P. David, J.-L. Dega, C. Rabéjac, H. Schindler, and J.-F. Soucailles, "Fault tolerant computing," in *Encyclopedia of Electrical and Electronic Engineering*, J. G. Webster, Ed. New York: Wiley-Interscience, 1999, vol. 7, pp. 285–313.

[9] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *Proc. 1st IEEE-CS Int. Computer Software and Applications Conf. (COMPSAC 77)* 1977, pp. 149–155.

[10] L. Chen and A. Avizienis, "N-version-programming: a fault-tolerance approach to reliability of software operation," in *Proc. 8th IEEE Int. Symp. Fault-Tolerant Computing (FTCS-8)* 1978, pp. 3–9.

[11] Y. Deswarte, K. Kanoun, and 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, and E. H. Sibley, Eds. Los Alamitos, CA: IEEE Comput. Sci. Press, 1999, pp. 171–182.

[12] Y. Deswarte, L. Blain, and J.-C. Fabre, "Intrusion tolerance in distributed systems," in *Proc. IEEE Symp. Security and Privacy* 1991, pp. 110–121.

[13] D. Powell, G. Bonn, D. Seaton, P. Veríssimo, and F. Waeselynck, "The Delta-4 approach to dependability in open distributed computing systems," in *Proc. 18th IEEE Int. Symp. Fault-Tolerant Computing Systems (FTCS-18)* 1988, pp. 246–251.

[14] D. Powell, A. Adelsbach, C. Cachin, S. Creese, M. Dacier, Y. Deswarte, T. McCutcheon, N. Neves, B. Pfitzmann, B. Randell, R. Stroud, P. Veríssimo, and M. Waidner, "MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications)," in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN'2001), Suppl.* 2001, pp. D32–D35.

[15] M. Backes and C. Cachin, "Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries," in *Proc. 2003 Int. Conf. Dependable Systems and Networks (DSN'2003)* pp. 37–46.

[16] M. Correia, L. C. Lung, N. F. Neves, and P. Verissimo, "Efficient byzantine-resilient reliable multicast on a hybrid failure model," in *Proc. 21st IEEE Symp. Reliable Distributed Systems (SRDS'2002)* 2002, pp. 2–11.

[17] M. Correia, P. Verissimo, and N. F. Neves, "The design of a COTS real-time distributed security kernel," in *Proc. 4th Eur. Dependable Computing Conf. (EDCC-4)* 2002, pp. 234–252.

[18] R. Stroud, I. Welch, J. Warne, and P. Ryan, "A qualititative analysis of the intrusion-tolerant capabilities of the MAFTIA architecture," presented at the Int. Conf. Dependable Systems and Networks (DSN 2004), Florence, Italy.

[19] C. Cachin, "Distributing trust on the Internet," presented at the Int. Conf. Dependable Systems and Networks (DSN-2001), Goteborg, Sweden.

[20] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proc. Recent Advances in Intrusion Detection (RAID 2001)* pp. 85–103.

[21] Y. Deswarte, N. Abghour, V. Nicomette, and D. Powell, "An intrusion-tolerant authorization scheme for internet applications," in *Proc. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN'2002), Suppl.* pp. C.1.1–C.1.6.

[22] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. E. Uribe, "An architecture for an adaptive intrusion-tolerant server," in *Security Protocols: 10th International Workshop*. Heidelberg, Germany: Springer-Verlag, 2004, vol. 2845, Lecture Notes in Computer Science, pp. 158–178.

[23] J. Fraga and D. Powell, "A fault and intrusion-tolerant file system," in *Proc. IFIP 3rd Int. Conf. Computer Security* 1985, pp. 203–218.

[24] J.-M. Fray, Y. Deswarte, and D. Powell, "Intrusion-tolerance using fine-grain fragmentation-scattering," in *Proc. IEEE Symp. Security and Privacy* 1986, pp. 194–201.

[25] M. O. Rabin, "Efficient dispersal of information for security, load balancing and fault tolerance," *J. ACM,* vol. 36, no. 2, pp. 335–348, 1989.

[26] L. Blain and Y. Deswarte, "An intrusion-tolerant security server for an open distributed system," in *Proc. Eur. Symp. Research in Computer Security* 1990, pp. 97–104.

[27] J.-C. Fabre, Y. Deswarte, and B. Randell, "Designing secure and reliable applications using FRS: an object-oriented approach," in *Proc. 1st Eur. Dependable Computing Conf. (EDCC-1)* 1994, pp. 21–38.

[28] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla, "Survivable information storage systems," *Computer,* vol. 33, no. 8, pp. 61–68, 2000.

[29] F. B. Schneider and M. A. Marsh, "CODEX: A robust and secure secret distribution syste," *IEEE Trans. Depend. Secure Comput.,* vol. 1, no. 1, pp. 34–47, Jan.–Mar. 2004.

[30] J. H. Lala*, OASIS—Foundations of Intrusion-Tolerant Systems*. Los Alamitos, CA: IEEE Comput. Sci. Press, 2004.

[31] C. Ko, T. Fraser, L. Badger, and D. Kilpatrick, "Detecting and countering system intrusions using software wrappers," in *Proc. 9th USENIX Security Symp.* 2000, pp. 145–156.

[32] T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal, R. Watro, M. Cukier, and J. Gossett, "Providing intrusion tolerance with ITUA," in *Proc. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN'2002), Suppl.* 2002, pp. C.5.1–C.5.3.

[33] L. Zhou, F. B. Schneider, and R. V. Renesse, "COCA: a secure distributed online certification authority," *ACM Trans. Comput. Syst.,* vol. 20, no. 4, pp. 329–368, 2002.

[34] P. Luenam and P. Liu, "The design of an adaptive intrusion tolerant database system," in *Proc. IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN'2002), Suppl.* 2002, pp. C.2.1–C.2.8.

[35] F. Wang, F. Jou, F. Gong, C. Sargor, K. Goseva-Popstojanova, and K. Trivedi, "SITAR: a scalable intrusion-tolerant architecture for distributed services," in *OASIS—Foundations of Intrusion-Tolerant Systems*, J. H. Lala, Ed. Los Alamitos, CA: IEEE Comput. Sci. Press, 2003, pp. 359–367.

Yves Deswarte (Member, IEEE) received the Certified Engineer degree from Institut Supérieur d'Électronique du Nord (ISEN), Lille, France, in 1972 and the Computer Science Specialization Engineer degree from Sup'Aéro, Toulouse, France, in 1973.

Formerly R&D engineer in a French computer manufacturer company, he joined INRIA and LAAS-CNRS, Toulouse, in 1979. He is currently *Directeur de Recherche* and member of the Dependable Computing and Fault Tolerance Research Group. He has been a major contributor to the design of many fault tolerant computing systems. His current research is devoted to intrusion tolerance, quantitative evaluation of security, security policies, and privacy preserving authorization schemes. He is the author of more than 80 international publications in the domains of fault tolerance and security in distributed systems. He is chairing the Société de l'Électricité, l'Électronique et les Technologies de l'Information et de la Communication (SEE) TC on Trustworthy Computing Systems and has chaired the Steering Committee of ESORICS, the European Symposium on Research in Computer Security, from its creation until 1998.

Mr. Deswarte is a member of the ACM Special Interest Group on Security, Audit, and Control (SIGSAC), the IEEE Computer Society (TC on Security and Privacy) and the Council of European Professional Informatics Societies (CEPIS) Special Interest Network on Legal and Security Issues. He is a senior member of SEE and member of SEE Board of Directors. He is representing the IEEE Computer Society at IFIP TC-11 on security and protection in information processing systems.

David Powell (Member, IEEE) received the B.S. degree in electronic engineering from the University of Southampton, Southampton, U.K., in 1972, a Specialty Doctorate degree from the Toulouse Paul Sabatier University, Toulouse, France, in 1975, and the *Docteur ès-Sciences* degree from the Toulouse National Polytechnic Institute in 1981.

He is currently *Directeur de Recherche* and a member of the Dependable Computing and Fault Tolerance Research Group at LAAS-CNRS. His research interests include dependability in the face of accidental and intentional human faults, fault-tolerant distributed systems, and dependability assessment. His current focus is ubiquitous computing and autonomous robot systems. He has authored or coauthored three books, 102 papers, and 17 book chapters, managed several national and European research contracts, and acted as a consultant for companies in France and for the European Commission. He was the Scientific Director of the Delta-4 FP3 Esprit project and the Scientific Advisor of the GUARDS FP4 Esprit project.

Dr. Powell is a member of ACM, SEE, and the IFIP 10.4 working group on dependable computing and fault tolerance. He served as program chair of the first European Dependable Computing Conference (Berlin, Germany, 1994), program cochair of the 26th IEEE Symposium on Fault Tolerant Computing (Sendai, Japan, 1996) and guest editor of a special section of *Communications of the ACM* devoted to group communication (April 1996).