Published in Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland (Ca.), May 20-22, 1991

Intrusion Tolerance in Distributed Computing Systems

Yves Deswarte Laurent Blain Jean-Charles Fabre

LAAS-CNRS and INRIA
7, avenue du Colonel Roche
31077 Toulouse (France)

Abstract

An intrusion-tolerant distributed system is a system which is designed so that any intrusion into a part of the system will not endanger confidentiality, integrity and availability. This approach is suitable for distributed systems, because distribution enables isolation of elements so that an intrusion gives physical access to only a part of the system. By intrusion, we mean not only computer break-ins by non-registered people, but also attempts by registered users to exceed or to abuse their privileges. In particular, possible malice of security administrators is taken into account. This paper describes how some functions of distributed systems can be designed to tolerate intrusions, in particular security functions such as user authentication and authorization, and application functions such as file management.

Introduction

Most of the currently developed secure systems are based on paradigms such as *access control matrix*, *reference monitor*, *security kernel* or *trusted computing base* concepts. These concepts are essentially centralized, in order to keep their implementation simple and verifiable. Such a centralized approach is inconsistent with distribution, local autonomy and concurrency that distributed systems are supposed to provide. Moreover, a centralized implementation of these concepts would constitute a "single point of failure", with respect both to accidental faults (a single site failure can produce a denial of service for the whole distributed system), and to intrusions (a successful intrusion into a single site is sufficient to annihilate the security of the whole distributed system).

For such distributed systems, the "Red Book" (Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria [15]) proposes the building of a Network Trusted Computing Base, composed of a set of cooperating Trusted Computing Bases. Within each site of the distributed system, a local TCB is responsible for the authentication of local users, and for the access control to local objects. For accesses from local subjects to remote objects, the local TCB must cooperate with the remote TCBs responsible for the objects. The enforcement of the authorization policy is based on cooperation between the TCBs, which must therefore trust each other, i.e. all the computers of the distributed system must enforce the same security concepts, with a consistent knowledge of subjects and objects, and with homogeneous security protocols. Consequently, this approach is unsuitable for current heterogeneous open distributed systems. Moreover, a successful intrusion into a local TCB can endanger the security of the whole distributed system. Such a case has to be seriously considered since, with current workstations, it is easy for a local user to obtain complete local control (e.g. as superuser). In addition, TCB administrators may be targets for bribery.

In our approach, the required trust comes from the cooperation and the consensus of a majority of security entities. Each entity can be individually untrusted, as long as a majority of them can be trusted. Therefore, an intrusion into a part of the system will have no consequence on the system security if only a minority of the security entities is affected by the intrusion : this approach is thus "intrusion-tolerant".

More precisely, let us consider a distributed system composed of standard workstations, the *clients*, and of intrusion-tolerant distributed *servers*. Each server is constituted by a set of untrusted sites, the server being trusted as a whole. That means that an intrusion into the distributed server sites should not endanger the confidentiality and integrity of the sensitive data stored or processed by the server, and should not produce any denial of service: to be successful, an attacker would have to intrude into a majority of the server sites, or bribe a majority of the site administrators. This

approach can be envisaged for application servers, such as file servers or data processing servers, as well as for specific security servers, responsible for user authentication and for authorization (i.e. control of access to application servers). The following parts of this paper will be devoted to the design of such servers.

The clients can be off-the-shelf workstations, but running only one user session at a time, and only for local users. Except during local user sessions, no sensitive data is stored on the workstation (temporary files and other session information are wiped out at the end of each session). No trust is placed on the workstation since all security relevant operations are run by intrusion tolerant servers. However, this paper does not address the threat of intrusions into the workstation during a user session, nor the problem of malicious logic (e.g. Trojan horses) inserted into a client or a server. Moreover, this paper does not consider input-output operations which are not located on the workstation: the intrusion tolerance approach does not seem suitable for printer servers or scanner servers, for instance. On the other hand, intrusion tolerance techniques can be applied to gateways and communication links [11, 20] or to data processing servers [8, 23], but such applications are beyond the scope of this paper.

The first section of this paper describes the intrusion tolerance approach more precisely, while the following sections are devoted to various particular intrusion tolerant servers: authentication servers, authorization and directory servers and persistent file servers.

1 Intrusion tolerance

By intrusion we mean a large class of attacks, covering not only computer break-ins by external attackers, but also illegitimate use by registered users [3]. Such intrusions can be classified according to the intruder privileges, or according to the intrusion targets. Intruders can be:

- *external intruders*, i.e. who are not registered as users of the computing system; thus, they have to deceive or by-pass the authentication and authorization mechanisms, or
- *internal intruders*, i.e. who are registered as legitimate users, but who:
  - try to exceed their privileges, for instance by trying to read confidential data or modify sensitive information for which they have no authorized access: to do this, they have to by-pass the authorization mechanisms, or
  - abuse their privileges for some illegitimate (but authorized) actions; for instance a security officer *can* (but *should not*) take malicious actions, or an operator *can* (but *should not*) halt a computer at some inappropriate instant, causing a denial of service[1].

Intrusion targets can be:

- *information confidentiality*: the intrusion attempts to disclose confidential information,
- *information integrity*: the intrusion attempts to create false information or to alter or destroy sensitive information,
- *service availability*: the intrusion attempts to prevent legitimate users from using the system (denial of service).

In *dependability* terminology [13], intrusions are *intentional operational external faults*, i.e. one particular class of faults. Classically, dependability is obtained by a mixture of fault prevention and fault tolerance. This dual approach can also be applied to intrusions, i.e. we can consider intrusion prevention and intrusion tolerance. Intrusion prevention is the aim of most of physical and logical access control mechanisms. When one considers the possibility that such mechanisms can be defeated, intrusion tolerance techniques can be contemplated.

As a matter of fact, intrusion tolerance is not a new concept. Cryptography, for instance, can be viewed as a very efficient technique for tolerating intrusions against data confidentiality as well as for detecting unauthorized data modifications (by means of cryptographic signatures). Moreover, classical fault tolerance techniques can be useful for tolerating intrusions: error detection-and-recovery or error masking techniques can be applied to maintain data integrity or service availability in spite of intrusions. However, such fault-tolerance techniques are usually considered as harmful for data confidentiality, due to the redundancy that they imply.

Our approach to intrusion-tolerance takes advantage of the geographic distribution of a distributed system in order to achieve confidentiality and integrity of sensitive information, and availability for the service. One of the techniques we propose for this purpose is the *fragmentation-redundancy-scattering* technique [6], which consists in first cutting all sensitive information into fragments, in such a way that any isolated

[1] Such intrusions are possible only because the *least privilege* principle is not perfectly implemented: otherwise, no illegitimate action would be authorized.

fragment does not contain significant information, and second scattering the fragments among the different sites of the distributed system, so that an intrusion into a part of the distributed system give access only to unrelated fragments. Redundancy is added to the fragments (e.g. by replication) in order to tolerate accidental or deliberate destruction or alteration of fragments. The following chapters show how this technique can be applied to different functions.

2 User registration and authentication

In our distributed system model, clients are untrusted, off-the-shelf workstations. Therefore, user authentication cannot rely on the workstation authentication mechanisms: user authentication has to be implemented by a trusted authentication server, which must also be responsible for user registration.

This approach is very similar to Kerberos [14] or Strongbox [24], except that these two authentication facilities are not intrusion-tolerant. For instance, the Kerberos server stores plain text passwords for all the users, which means that if an intrusion succeeds on the server, the intruder can use this information to impersonate any user. Strongbox uses zero-knowledge protocols, and then disclosure of information stored on the authentication server is not sufficient for an intruder to masquerade other users. However, such masquerade is possible with the complicity of one of the authentication server administrators: an administrator can modify any user registration information.

2.1 Intrusion tolerance for user registration and authentication

An intrusion-tolerant registration and authentication server must tolerate all the kinds of intrusions which have been presented in section 1. This can be done by the use of distribution among a set of authentication sites. Thus, availability of the authentication function can be achieved in spite of failures of one or a minority of sites. Another characteristic is that each site can be managed by a different security administrator: this enables intrusions by a minority of malicious security administrators to be tolerated. Thus, the authentication server can be viewed as composed of a trusted set of untrusted authentication sites managed by a trusted set of untrusted security administrators.

2.1.1 User registration

Registration of a new user in the system must be performed under the control of the security administrator at each authentication site. The user is registered at the first site by the first security administrator, at the second site by the second administrator and so on. At each site, the identity of the user is stored with some authenticator which will be used by the authentication process to verify the claimed identity. The authenticator can be a password, a secret permanent key or a public key. However, except if public keys are used, different authenticators have to be stored on the different authentication sites for the same user. This can be considered as an implementation of the concept of *separation of duty*: one or a minority of security administrators cannot register an illegitimate user and cannot prevent the registration of a legitimate user at a majority of authentication sites. Moreover, if such malicious administrators try to use local information stored on their sites in order to impersonate a registered user, they will fail because they cannot be authenticated by the other sites.

2.1.2 Authentication protocol

When a registered user wants to access remote servers from a workstation, he/she has to be authenticated by the authentication server: an authentication protocol has to be run between the user site and the authentication sites. This protocol is composed of three phases. In the first phase, the user site attempts an independent authentication with each of the authentication sites. This authentication can be based on classical authentication algorithms [16, 12, ...], but with different authenticators (or different challenges with public key systems) for the different authentication sites. In this first phase, each authentication site independently decides, for itself, whether or not the authentication attempt succeeds. During the second phase, each authentication site broadcasts its individual decision to all the other authentication sites, and receives the decisions of these other sites. In the third phase, according to the majority of all the received decisions and its own decision, the authentication site authorizes (or not) the session for the user, and confirms this session authorization by sending to the user site a session key (or a ticket containing the session key, depending on the authentication algorithm which has been selected) . A different session key is randomly generated by each authentication site. This session key or ticket will be used by the user site to authenticate its requests in the authorization process (see section 3).

In this protocol, majority voting on the different authentication decisions enables the system to tolerate accidental faults affecting the registration data stored

on the different authentication sites or communication faults during the protocol, as well as to tolerate intrusions into a minority of authentication sites which could lead to false local decisions. Moreover, different session keys are generated independently by each authentication site, and sent only to the user site; thus, no intruder, even with the complicity of an administrator, can impersonate the user site (except if he/she breaks the classical authentication algorithms).

To conclude this part, we can say that the authentication server described above has two essential characteristics. A read intrusion into one or into a minority of authentication sites does not give enough information to impersonate a user and the destruction of data of one or of a minority of sites does not make the authentication function unavailable.

2.2 Implementation of the registration and authentication protocols

It would be difficult for a user to memorize several strong independent passwords. A better solution is to store secret keys on a personal smartcard, the owner of which has only to memorize his/her PIN. Our current implementation uses Bull CP8 smartcards with shared keys, one smartcard for each user and one for each authentication site administrator; all the administrators of a given authentication site have identical master smartcards, except for the identification and the PIN of the administrator. On the user smartcard, there is a set of areas, one area for each authentication site, i.e. for each master smartcard. This means that when a user is registered by an authentication site, the local master smartcard generates a secret key that it writes within its own reserved area on the user smartcard. When the user has been registered by the N sites, his/her smartcard possesses N secret keys within N areas. In the authentication phase, each authentication site sends a challenge to the user. The user smartcard applies a one-way function to this challenge and the shared secret key and sends the result to the authentication site. The master smartcard performs the same operations on the same data and compares the results. This protocol is performed by every authentication site.

The major drawback of this system is that, when a new authentication site is added, all users must be registered by the new site. Another solution for implementation of these functions is the use of public-key cryptosystems such as RSA [18] or El-Gamal [4] or zero-knowledge protocols [5]. The interesting point of these systems is that they do not need shared secrets. Only the user (or the user smartcard) knows the secret and there is no risk of masquerade by security administrators. Another advantage is that when a new authentication site is added, it can immediately perform user authentication because it has only to copy locally the public keys which are stored on the other authentication sites. Unfortunately, currently there is no efficient public-key smartcard available.

3 Authorization and directory servers

The aim of this section is to describe a distributed, intrusion-tolerant authorization server. This server has to store and manage access-rights and to grant or deny user accesses to remote application servers. The server is made intrusion-tolerant using distribution among a set of authorization sites and the application of the following techniques: replication, secret sharing and agreement.

3.1 Distributed system authorization

An authorization server has to *check* the rights of users who wish to access remote servers or objects. It also *manages* all user rights for all objects which could be accessed and which need to be protected. When a new object is created by a user or when a new server is installed, the authorization server stores the access rights, the reference (information which enables direct access to the object) and other information, which together form the object or server descriptor. The access right management must obey an *authorization policy*, which has to be implemented by the authorization server.

The two roles of an authorization server are given above: access right checking and access right management according to an authorization policy. In most secure distributed systems, these two roles are performed by the secured application servers. For instance, in the Kerberos system [14, 22] an authentication server and a ticket granting server verify the identity of a user who wishes to access a remote application server and establish the session between the user site and the application server, but the authorization is carried out by the application server itself: once a session has been established between the user site and the application server, the authentication server and the ticket granting server play no further role, and the application server is alone responsible for access right checking as well as for access right management.

The Kerberos approach possesses the advantage of incremental modularity: it is easy to add a new server

in the system, just by adding server identification information to the ticket-granting server database. This approach also permits flexibility of authorization management: each server administrator manages access rights the way he/she wishes to, with some local authorization policy he/she selects. The price paid for this flexibility is the necessity to develop specific authorization mechanisms for each server, and to administrate individually each server. Moreover, the consistency of the distributed system authorization policy and the consistency of the administration of the whole distributed system are based upon the cooperation and the benevolence of server administrators.

Another approach is to implement a central reference monitor on a specialized site, such as for the Secure File System of the Distributed Secure System architecture proposed in [19]. In that case, the authorization policy consistency is easily enforced, and the security administration of the distributed system is very simple. But, since the reference monitor is the mediator for every communication between the user site and the server, this reference monitor site is a bottle-neck for the communications and a single point of failure.

Our approach can be considered as a compromise between these two approaches: an authorization server is responsible for access right checking and access right management, but when an access from a user site to an object is granted, tickets are distributed by the authorization server to the user site and to the server managing the object, so that the user site can directly access the server for all its requests to that object: when the application server receives a request, it has only to check that this request is allowed by the corresponding ticket. An application of this protocol is presented in section 4. This kind of authorization server gives the same authorization policy consistency and administration simplicity as the central reference monitor, but with no communication bottle-neck. Section 3.3 shows how to make this authorization server intrusion-tolerant and fault-tolerant in order to avoid it constituting a single point of failure.

3.2 Integration of directory and authorization functions

The descriptors (access-rights, references,...) for all the objects are stored on the authorization server; thus it is necessary to have a representation and a request model for these descriptors. The two problems of concern here are how to store and how to access these descriptors. A very convenient model is the Directory Service standard [2]. The data representation given in this standard is a tree where all the leaves are descriptors and the nodes are directories [9]. The tree model permits the building of a hierarchy of objects. For each node or leaf of the tree, there is a set of attributes which describes the object, the server or the directory. According to the object type, the attributes are different; this facilitates the representation of the various types of servers which exist in the system.

The authorization server has to manage two kinds of access controls. The first concerns access to application servers, such as a data processing servers, which manage no specific persistent objects. In the tree structure, the descriptors of such servers are leaves directly connected to the root: there is no obvious reason to structure these servers as a hierarchy. The second kind concerns accesses to persistent objects maintained by application servers, such as the Persistent File Server presented in section 4. In this case, access control concerns not only the servers but also the objects which are stored inside (files in this example). The representation of these servers and their objects will be a subtree, the root of which will be under the global root. This tree has both nodes and leaves. The nodes are object directories and do not correspond to real objects; they are only means for managing a hierarchy of objects. The leaves are descriptors of real objects and keep all the information needed for access.

If the distributed system is large enough, all the servers and objects may not be represented in a single authorization server. In such a case, the distributed system can be partitioned into multiple management domains, with an authorization server in each domain. In this case, the tree of objects and servers managed by an authorization server is represented in other authorization servers by a link from a node to the root of this tree (figure 1).

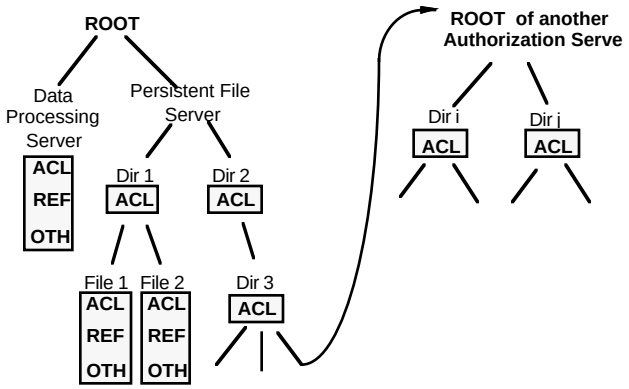

Figure 1: Example of directory structure.

Within the tree, the leaves are descriptors of real objects or application servers which can be accessed. For each leaf, there is a set of attributes which compose together all the information about the corresponding objects (or application servers). These attributes are:

- access control list, which defines which access-rights to the object are granted and to whom;
- reference, which is information which enables direct addressing of the object or server;
- other information according to the object type.

For directories, except for links to other authorization servers, there is no reference attribute because they do not match a real object in the system. The only attribute for directories is an access-control list.

Once created this data must be accessed by users, as for any classical Directory Service, by means of requests to the authorization server (X500 requests such as descriptor read, modification, adding and deleting). An important point is that when the authorization server receives the request, it acts according to the authorization policy which has been implemented. The authorization policy does not depend on the representation model, so that any type of policy can be implemented (multi-level, discretionary...). Nevertheless, if a mandatory policy is selected, then this policy must be enforced within the user site, which is not realistic with current off-the-shelf workstations. However, a discretionary authorization policy, compatible with these requirements, has been proposed in [1].

3.3 Intrusion-tolerant implementation

The problem raised in the above section is "how to make a centralized authorization server intrusion-tolerant ?". This raises the question of "how to store the objects descriptors on the server" and "how to run the server" so that intrusions have no consequence on the information confidentiality and integrity and on the service availability. The answer is distribution, data replication, secret sharing and majority voting.

3.3.1 Distribution

First of all, just as with the authentication server, the authorization server is distributed among several sites called authorization sites. Each authorization site is administrated by a different security administrator. No individual site or administrator is trusted, although a majority of them are trusted.

The authorization server is strongly related to the authentication server since only users that have been previously authenticated can access remote servers. At least for small distributed systems, the simplest solution is to locate the two servers on the same set of sites. But it is also possible to isolate authentication sites (which have to store only user registration information) from authorization sites (which have to store only server and object information): in such a case, the authentication server manages a directory tree consisting of just a link to an authorization server, i.e. the relation between the authentication server and the authorization server is the same as the link from one authorization server to another one.

3.3.2 Data replication and secret sharing

Among all the information stored by the directory-and-authorization servers, some data items are less sensitive than others. For instance, one can consider that the tree structure and the access control lists are not confidential enough to prevent security administrators from reading them, the integrity and availability of these data items being much more important than their confidentiality: such data can be replicated on all the authorization sites[1].

Other data items are more sensitive, since they could be used to by-pass or deceive the access controls: that is the case of permanent keys, such as the fragmentation key used by the persistent file facility (see section 4). These very confidential data items are object attributes which must be managed in such a way that only an agreement between a majority of the authorization sites can give access to them: such

[1] If a subtree structure belongs to a confidentiality level higher than the rest of the tree, this subtree can be isolated in another authorization server, operated by other, more trusted security administrators, with a link from the main tree to the subtree (see figure 1).

secrets are thus to be "shared" [21] by the authorization sites. A very satisfactory solution to this problem is given by the "threshold scheme" concept: a threshold scheme is used on the user site to build different "shadows" of the confidential information. Then, these shadows are sent to all the authorization sites. The local attribute stored at a single authorization site is only a part of the overall attribute of the object; this attribute can be only rebuilt on the user site which gets a sufficient number of shadows (greater or equal to the threshold, which can be for instance the majority of authorization sites). This means that a minority of authorization sites cannot use this attribute to perform illegal operations. Only the authorized user site can rebuild the confidential data when it receives a majority of correct shadows.

Finally, a third kind of confidential data is managed by authorization sites: user session keys, server keys, etc. But these data items are different on each authorization site: security is maintained as long as only a minority of these data items are disclosed.

3.3.3 Majority voting and authorization protocol

The authorization protocol is quite similar to the authentication protocol: whatever the user request, first a local decision is taken by each authorization site according to the user access rights which are locally stored; then this local decision is broadcast to the other authorization sites; the authorization decisions received from the other sites are voted on locally (together with the local decision), and, according to the result of the vote, the user request is locally executed or not. This majority voting technique ensures that a legitimate request cannot be denied and an illegitimate request cannot be granted, unless a majority of authorization data copies have been destroyed or altered.

In fact, two kinds of user requests have to be considered. The first one concerns only the authorization and directory sites, such as directory read or modification, or access control list modifications, etc. In such cases, the authorization process is limited to the local execution of the user request at each authorization site. In other cases, the user request is to access an object or an application server which is not located on the authorization sites. For that purpose, after majority voting, if access is authorized, each authorization site will send one ticket to the user site and another to the application server.

The user site ticket is quite similar to the ECMA PAC (Privilege Attribute Certificate) defined in [10]. The ticket contains tamperproof information concerning the operations which are allowed on the object or on the server. It also contains the object or server reference, but this reference is only transmitted by means of different "shadows" (see section 3.3.2) from each authorization site: only the user site which receives the different tickets from a majority of authorization sites can rebuild the effective reference. This means that no other site can impersonate the user, even with the complicity of the administrators of a minority of authorization sites.

When the user site has rebuilt the reference, it can reconstitute the access ticket containing the tamperproof access information, the complete reference and a user identity certificate. This ticket is valid for several accesses as long as they concern the same object or server with the same authorized operations (e.g. all read accesses for one persistent file, see section 4.2.2.). This ticket is sent by the user site to the application server with each request. The application server has only to check the validity of this ticket according to the majority of tickets it has received from the authorization sites, and to verify if the user request and the reference are valid, before executing the request.

4 Persistent file server

4.1 Overall framework

The aim of this section is to give an example of an intrusion-tolerant application server that takes advantage of authorization servers to simplify security management and administration. The application server presented here is a persistent file server, i.e. a server which stores the user files between user sessions [6, 7]. In our distributed system model, a typical user session consists of the following:

- the user starts his/her workstation (from which all user data have been wiped out at the end of the previous session) and runs the authentication protocol with the authentication server,
- the user reads a particular file from the persistent file server (if read access is granted by the authorization server),
- the user modifies the local file copy on his/her workstation,
- at the end of the session, all file updates are sent to the persistent file server (if write access is granted by the authorization server), and all local user data items are deleted.

Two types of intrusions have to be considered. First, confidentiality can be attacked by tapping the communication medium or by intruding into the file server. Second, an intruder can attempt to modify or to destroy a file on the file server (integrity and availability attacks). In addition, accidental faults can also endanger file integrity and availability. To deal with these threats, we propose to use the fragmentation-redundancy-scattering technique which has been defined in section 1.

The fragmentation and scattering technique when applied to file storage involves cutting every sensitive file into several fragments in such a way that one or several fragments (but not all) are insufficient to reconstitute the file. The level of information granularity is such that the contents of one or several fragments together do not disclose any significant information. The fragments are stored in several copies on different geographically distributed sites, which can be viewed as fragment server machines (see figure 2).

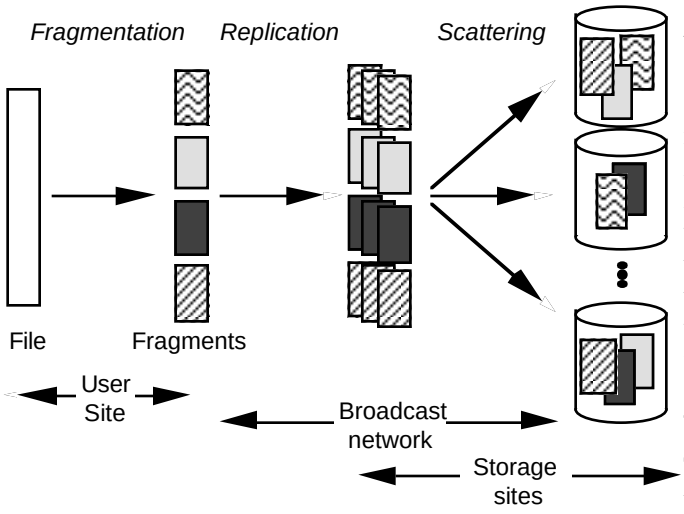

Figure 2: Fragmentation-and-scattering applied to persistent file storage

4.2 The file server structure

A persistent file is processed using a set of operations as described below. Some of the file management operations are carried out in the user site, others are remotely executed by the storage sites. To ensure a high level of security, whole files are never available except on the user site during the user session. Thus, data items transmitted by the network are always in a fragmented form, with no possibility of identifying the different fragments belonging to a given file (otherwise, eavesdropping of the communication channel could annihilate the added advantages of this technique). That means that the fragmentation and naming of fragments are executed in the user site.

4.2.1 User site operations

Every user site is able to access the persistent file operations by means of specific library functions for storing and retrieving the files within the distributed file system. These user site operations use the fragment server functions of the different storage sites and the directory functions of the authorization server. The library provides all the usual file operations such as creation, deletion, opening, closing, reading, writing, etc.

As stated earlier, two basic operations related to file security are provided in the user site: fragmentation and fragment naming. The fragmentation operation uses a fragmentation key which is stored and distributed by the security server by means of a threshold scheme. The fragmentation operation is based on fast and simple algorithms that give flexible access to any file whilst ensuring a high level of security due to the scattering of information. The names given to the different fragments are generated by cryptographic methods using the fragmentation key, such that no information can be derived from these names. Naming is carried out in such a way that fragments have a unique identifier, derived from the fragmentation key, the name of the file and some other parameters. During the read operation, the original file is reconstituted by using a similar algorithm and the same key as for fragmentation. Section 4.3 details these operations.

4.2.2 Fragment server site operations

The persistent file server relies on fragment server functions provided by a set of storage sites. The operations of these sites correspond to simple space allocations on the physical storage devices and data transfers between the storage device and the network. These fragment operations are only available to the file management operations embedded in the user sites, and only if an authorization server has sent the corresponding ticket (see 4.2.3). When writing a file, the user site sends the fragments over the communication channel to the set of fragment server sites. For any fragment, each site decides whether or not it should be stored locally, depending on a distributed algorithm ensuring security and availability, based on principles discussed in section 4.4. For the read operation, the user site broadcasts the names of the fragments, and for each

fragment, every storage site which had stored a fragment copy sends it to the user site.

Each storage site acts as a file server which can only store fixed length files, with a "flat directory" structure. The operations managed by these storage sites are fragment reading, fragment writing, and fragment deletion. Only fragment names are visible at any storage site; thus, an intruder who has obtained the control of a storage site cannot determine where a fragment comes from or to which file a fragment belongs.

4.2.3 Access control (authorization)

As has been described above, authorization is not performed by the storage sites but by an authorization server. The authorization server manages and verifies rights for files and not for the fragments because it knows neither their names nor their location. For each file, the directory function of the authorization server stores a reference of the file, which is in fact the fragmentation key which permits generation of the fragment names. The fragmentation key is "globally shared" and each authorization site has only one shadow of the key. When a user site rebuilds the key, it is able to access the fragments directly.

As described in section 3.3.3, the user site and the storage sites receive different tickets. The storage site ticket cannot contain the file name because the storage sites cannot know which file a request for a fragment corresponds to; this ticket cannot either contain the reference (fragmentation key) which concerns only the user site. Therefore, how does a storage site know that a fragment corresponds to a file that a user has been authorized to access? The solution is to store a tag within the fragment which is the result of a hash function applied to the fragment name and the file name. This tag is generated by the user site during the fragment write operation. The storage site ticket contains the hashed file name generated by the authorization server. The storage site, when receiving a request for a fragment, applies the hash function to the fragment name and the hashed file name, and compares the result with the fragment tag: if they are identical, this means that the fragment belongs to the file the user is authorized to access. The last field of the storage site ticket corresponds to the authorized operations and to the user identity certificate; these data items are compared to the ticket the user site transmits with its request.

4.3 Fragmentation principles

A general approach is proposed for the fragmentation operation. While, the file may be of any length and of any type, the fragmentation operation must ensure that no useful information can be obtained from isolated fragments, which implies that all the fragments (from all the files) must have the same fixed length and that their names do not allow any information to be deduced. Finally, an important requirement concerns data integrity: modification of a fragment must be easily detected when the file is read.

4.3.1 Partitioning

A method has to be defined that is suitable for producing fragments of identical length from files with very different lengths. The solution proposed is first to cut each file into pages of fixed size (partitioning). The files are padded out to reach a size equal to a multiple of a page size (figure 3).

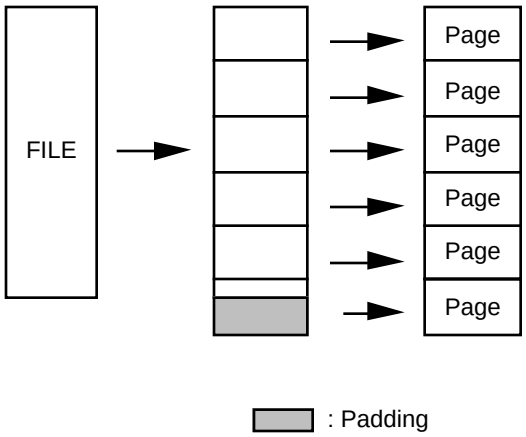

Figure 3: A file partitioned into pages

Every page may then be fragmented into an identical number of fragments. All the fragments so obtained have the same length, which may be chosen equal to that of a packet sent on the communication channel, or a quantum in the mass storage, for example, in order to improve the speed of access to information. Another advantage is that one does not need to get the whole file: pages can be retrieved independently. So, a user does not need to reassemble a whole file if he/she only needs a single page. A cryptographic checksum is added to each page; this checksum is checked by the read operation to verify the integrity of the page.

The mean space overhead due to padding information is half a page (if the mean file length is much larger

than the page length), and is of course a large overhead for very small files. The shorter a page is, the smaller this overhead is, but the longer is the management time, mainly due to fragment storage time.

4.3.2 Fragmentation

Fragmentation is performed at the user site and is realized in conjunction with classical cryptographic methods. One may wonder why fragmentation is used since cryptographic techniques must still be employed. There are two good reasons:

- first, the geographical scattering of fragments makes the theft of individual storage media of no avail to the intruder - even if he/she possesses the cypher key,
- secondly, the added security of scattering means that the ciphers employed can be much simpler and thus faster than conventional ones.

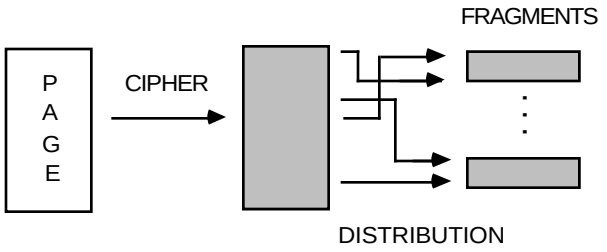

Figure 4: Ciphering and fragmentation

We have thus chosen to fragment ciphered pages (figure 4): each page is first ciphered and the fragments are obtained from this ciphered page. The distribution uses a fixed scheme wherein each successive quantum of data is put into one of the fragments; this distribution does not depend on the key. This operation leads to a fine-grain scattering of the data among all the fragments.

In order to make it as difficult as possible for an intruder to decipher an individual fragment or sub-set of fragments, it is preferable to choose a cipher scheme that makes the ciphertext of each data-quantum, and thus each fragment, dependent on the others. This may be realized by using a stream cipher, for instance by means of *cipher block chaining* (CBC). In order to make the cryptanalysis more difficult (even if the intruder is eavesdropping the network and is observing several versions of the same fragments), a random nonce (generated each time a page is written) is added at the beginning of the plain text page. Once the ciphered block is obtained, regular fragmentation is carried out: fragment number *j* contains the fixed quantum (bits, bytes, words...) number *i* such that j=i (mod N), *N* being the number of fragments.

Fragment naming consists in assigning a unique identifier to every fragment; this unique identifier is derived from the fragmentation key, the name of the file, the index of the page and the index of the fragment. The naming algorithm is based on one-way cryptographic functions such that no information concerning a fragment can be derived from its name.

The fragment writing, reading and deletion requests which are transmitted by the user site to the storage sites are sent in a random order for each page. That means that if an intruder is eavesdropping the network or controls a storage site, he/she can receive all the fragments of a given page, but he/she is not able to know in which order he/she has to put the fragments in order to attempt a cryptanalysis of the page. For instance, if a page is cut in 16 fragments, an intruder should attempt 16! /2 $\approx 10^{13}$ trials to find the correct arrangement. Thus, the confidentiality depends more on this random order than on the efficiency of the cipher.

4.4 Scattering principles

Once the file is fragmented at the user site, the fragments are broadcast to the storage sites in order to be stored with a fixed number of copies (i.e. on a fixed number of different storage sites) in order to ensure availability via replication; the number of copies is a file parameter which depends on the availability requirements of the user for that file. Using a broadcast communication channel, each fragment is sent only once and is, in general, received by all the storage sites.

Once a fragment has been received by the storage sites, a decision has to be taken by these sites in order to ensure that (exactly) R copies will be stored. A distributed pseudo-random algorithm is then executed that takes into account the relative available space at each site in order to decide the final placements of the fragments. The need to take into account the available space at each site is necessary in order to maintain a good balance among the different fragment server sites. On the other hand, (pseudo—)random behaviour is advantageous for preventing an intruder from knowing the actual locations of the replicates.

Scattering increases confidentiality because a number of concerted intrusions is necessary in order to get all the fragments derived from a single page. It also increases data availability, due to the replication of fragments. An intruder would have to destroy as many

sites as the number of replicates to make a fragment unavailable. Integrity properties are provided because an intruder would have to modify all the replicates and so must realize several intrusions. Moreover, it is very unlikely that an intruder could modify even one byte of a fragment without modifying the cryptographic checksum of the page.

4.5 Comparison to other related techniques

Other techniques could be used to implement a secure persistent file server. The first one consists in ciphering the file on the user site, and storing the ciphered file in several copies on different file servers. In that case, confidentiality relies on the efficiency of the cipher algorithm: an intruder who is eavesdropping the network or who gets a copy of the ciphered file (e.g. a file server back-up tape) can take all the time and all the computer power he/she wants to cryptanalyze the file; with one intrusion, he/she gets all the information he/she needs. With our technique, the intruder who gets all the fragments of a page, would have to try, say, 10^{13} arrangements before cryptanalyzing the ciphered page (cf. section 4.3.2). That means that our cipher can be $\approx 10^{13}$ times less strong, and can be much faster. For integrity, the two techniques are comparable. For availability, the two techniques are equivalent for accidental server failures, but the fragmentation-replication-scattering technique is less robust against simultaneous destruction of storage sites: if an intruder is able to destroy R (out of N) storage sites at the same time, he/she will make many more files unavailable than if he/she destroys R ciphered file servers. The overhead of the two techniques are equivalent for the communications and the storage space, but fragmentation-replication-scattering can be made much less CPU-consuming than the ciphered file approach.

Another technique has been proposed by Rabin for fault-tolerant file servers: the "Information Dispersal" approach [17]. This technique consists in coding the file with a special error-correcting code and in splitting the coded file in *n* pieces, such that *m* out of *n* of these pieces are sufficient to rebuild the complete file. The *n* file pieces are stored by different storage servers. The coded file is longer than the original file, but the redundancy is much smaller than replication in $(n - m + 1)$ copies, for nearly the same availability and integrity. But this code is much more CPU consuming than the fragmentation and does not ensure file confidentiality: to prevent information disclosure to eavesdroppers and storage site intruders, the file has to be ciphered before coding. To summarize, the storage and communication overhead is much less in the Rabin's technique, but it consumes much more CPU time.

Conclusion

The intrusion tolerance approach looks very promising for open distributed systems whose elements cannot be all trusted. In particular, the intrusion-tolerant authentication and authorization servers enables a consistent security policy to be implemented on a set of heterogeneous, untrusted sites, administrated by untrusted (but non-conspiring) people.

A prototype of the persistent file server presented in this paper has been successfully developed and implemented as part of the Delta-4 project of the European ESPRIT programme. An intrusion-tolerant security server, gathering the authentication function and the directory and authorization function, is currently being developed for the Delta-4 project; completion of a prototype is planned for the end of 1991. In parallel, application of the fragmentation-redundancy-scattering technique to data processing is explored [8, 23]: in that case, process parallelism and distribution are used to prevent information disclosure, while process replication is used to ensure availability and integrity.

Acknowledgements

The authors are grateful to Jean-Claude Laprie, David Powell and Joni Fraga for the original principles of the fragmentation-redundancy-scattering technique as well as to Jean-Michel Fray, Pierre-Guy Ranéa and Gilles Trouessin for their contribution to the development of this technique. Brian Randell has helped to make the final version of this paper more readable. This research is partly supported by the PDCS (Predictably Dependable Computing Systems) project n°3092 of the European ESPRIT programme, while some aspects have been implemented as part of the Delta-4 (Definition and Design of an open Dependable Distributed architecture) project n°2252 of ESPRIT.

References

[1] Blain L. and Deswarte Y., "An intrusion-tolerant security server for an open distributed system", *Proceedings of the European Symposium in Computer Security (ESORICS 90)*, Toulouse (France), October 1990, Pub. AFCET, ISBN 2-9036778-9, pp. 97-104

[2] CCITT, *The Directory*, Recommendation X500, December 88.

[3] Denning D.E., "An intrusion-detection model", *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, Oakland (Ca.), April 1986, pp. 118-131

[4] El Gamal T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", *IEEE Transactions on Information Theory*, Vol.31, n°4, July 1985, pp. 469-472.

[5] Fiat A. and Shamir A., "How to prove yourself: Practical solutions of Identification and signature Problems", *Advances in Cryptology - CRYPTO 86*, Santa Barbara (Ca.), August 1986, Springer-Verlag, Vol. 263, ISBN 0-387-18047-8, pp. 186-194.

[6] Da Silva Fraga J. and Powell D., "A Fault- and Intrusion-Tolerant File System", *Proceedings of the 3rd International Conference on Computer Security, IFIP/SEC'85*, Dublin (Ireland), August 1985, pp. 203-218.

[7] Fray J.M., Deswarte Y. and Powell D., "Intrusion-tolerance using fine-grain fragmentation-scattering", *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, Oakland (Ca.), April 1986, pp. 194-201.

[8] Fray J.M. and Fabre J.C., "Fragmented Data Processing: an Approach to Secure and Reliable Processing in Distributed Computing Systems", *Proceedings of the 1st IFIP International Conference on Dependable Computing for Critical Applications (DCCA)*, Santa Barbara (Ca.), August 1989, pp. 131-137.

[9] Gasser M., Goldstein A., Kaufman C. and Lampson B., "The Digital Distributed System Security Architecture", *Proceedings of NCSC*, 1989.

[10] Hoffmann G., Lechner S., Leclerc M. and Steiner F., "Authentication and Access Control in a Distributed System", *Proceedings of the European Symposium in Computer Security (ESORICS 90)*, Toulouse (France), October 1990, Pub. AFCET, ISBN 2-9036778-9, pp. 71-84

[11] Koga Y., Fukushima E. and Yoshirara K., "Error recoverable and securable data communication for computer network", *Proceedings of the 12th International Symposium on Fault-Tolerant Computing (FTCS-12)*, IEEE, Santa-Monica (Ca.), June 1982, pp. 183-186

[12] Lamport L., "Password authentication with insecure communication", *Communications of the ACM*, Vol. 24, n°11, November 1981, pp. 770-772

[13] Laprie J.C., "Dependability: a unifying concept for reliable computing and fault-tolerance", in *Dependability of Resilient Computers*, BSP Professional Books, Oxford (UK), Ed. T. Anderson, 1989, pp.1-28

[14] Miller S.P., Neuman B.C., Schiller J.I. and Saltzer J.H., "*Kerberos Authentication and Authorization System*", MIT Project Athena Technical Plan, Sect. E.2.1, December 1987.

[15] NCSC, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, Tech. Rept. NCSC-TG-005, National Computer Security Center, 31 July 1987.

[16] Needham R.M. and Schroeder M.D., "Using encryption for authentication in large networks of computers", *Communications of the ACM*, Vol.21, n°12, December 1978, pp. 993-999

[17] Rabin M.O., "Efficient dispersion of information for security, load balancing and fault tolerance", *Journal of the ACM*, Vol. 36, n°2, April 1989, pp. 335-348

[18] Rivest R. L., Shamir A. and Adleman L., "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, Vol.21, n°2, February 1978, pp. 120-126

[19] Rushby J.M. and Randell B., "A distributed secure system", *IEEE Computer*, Vol.16, n°7, July 1983, pp. 55-67

[20] Rutledge L.S., *A spatial encoding mechanism for network security*, Ph.D. dissertation, Institute for Information Science and Technology, Washington, March 1987

[21] Shamir A., "How to Share a Secret", *Communications of the ACM*, Vol.22, n°11, November 1979, pp. 612-613.

[22] Steiner J. G., Neuman C. and Schiller J.I., "Kerberos: An Authentication Service for Open Network Systems", *Proceedings of the USENIX Winter Conference*, Dallas (Texas), February 1988.

[23] Trouessin G., Fabre J.C. and Deswarte Y., "Reliable processing of confidential information", *Proceedings of the 7th International Conference on Computer Security, IFIP/SEC'91*, Brighton (UK), 15-17 May 1991.

[24] Yee B. S., Tygar J. D. and Spector A. Z., "*Strongbox: A Self-Securing Protection System for Distributed Programs*", Technical Report CMU-CS-87-184, January 1988, 18 p.