A Fault-Tolerant Secure CORBA Store using Fragmentation-Redundancy-Scattering

Cristina Silva
FC/UL
(csilva@navigators.di.fc.ul.pt)

Luís Rodrigues
FC/UL
(ler@di.fc.ul.pt)

Abstract

This paper presents the design of a secure and fault-tolerant CORBA datastore based on the Fragmentation-Redundancy-Scattering (FRS) technique. This technique consists in fragmenting the confidential data and scattering the resulting fragments across several archives. The FRS-Datastore service interacts with the other CORBA services, in particular with the Persistence, Security and Trading services. One of our goals is to gain a better understanding how the FRS technique can be applied to an open environment prone to crashes and network partitions and using exclusively standard invocations.

1 Introduction

The Fragmentation-Redundancy-Scattering (FRS) is a technique that can be used to achieve security and fault-tolerance [1]. It consists in fragmenting the confidential data and scattering the resulting fragments across several archive sites of the distributed system. Fragmentation is performed so that any isolated fragment contains no significant information. Scattering is performed in such a way that each archive contains just a subset of (unrelated) fragments and that each fragment is stored in more than one site. The technique provides intrusion-tolerance: if a node is compromised, the intruder has no access to relevant information (and if compromised fragments are deleted or altered, they can be recovered from other nodes).

This paper presents the design of a secure and fault-tolerant CORBA [2] datastore based on the FRS technique. Our aim is to implement a datastore that can be used by a Persistent Data Service [3] to securely store the state of Persistent CORBA objects. The FRS-Datastore design is based on the composition of CORBA objects which interact exclusively using standard object invocations. Thus, the FRS-Datastore can be deployed across different ORBs.

2 Fragmentation-Redundancy-Scattering

For self-containment we present a brief description of how the FRS technique can be applied to implement a secure and fault-tolerant store. This description is based on the implementation of the secure data archive of the Delta-4 system [4]. We will later highlight the main differences between our design and the design used in the Delta-4 system.

The basis of the technique is to cut the data in several fragments. The fragmentation operation must ensure that, once the fragments are isolated, no information can be

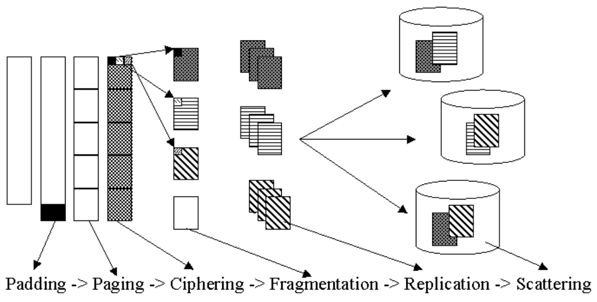
Padding -> Paging -> Ciphering -> Fragmentation -> Replication -> Scattering

Figure 1: FRS steps

obtained from them. Thus, the fragment size must not depend on the size of the original data. The idea is to split the data to be archived into pages of fixed size (adding some padding to the last page if necessary). Each page is ciphered and signed. The data of each page is then scattered among a fixed number of fragments. This procedure is illustrated in Figure 1. Fragments are named using a one-way cryptographic function and are derived from the object's persistent identifier, the fragmentation key, the page number, and the fragment number.

Once all fragments are produced, these are sent in random order to the archive nodes. The distribution policy must ensure that the fragments are distributed among the different archive nodes and that R copies of the same fragment are stored (for fault-tolerance). It should also promote some sort of load-balancing (i.e., to take into account the available space at each node). The service requires the availability of a security server where the fragmentation key for each object is securely stored (the implementation of the security server lies outside the scope of the paper).

The FRS technique complements the security provided by ciphering by making the store less prone to cryptanalysis: an intruder attacking an individual archive has no access to all fragments. Even if he/she gets access to all N fragments, $N!/2$ cryptanalysis have to be performed to re-constitute the clear page.

3 Design overview

The design of CORBA FRS-Datastore uses two types of objects: Mediators and Fragment Archives. The Mediators are responsible for the fragmentation and scattering of data, and are accessed through a Persistent Data Service (PDS). The Archives are responsible for storing individual fragments, and export a private interface to the Mediators. In some cases, the Archives will issue call-backs to the Mediators, thus Mediators and Archives assume both the role of clients and servers with regard to each other.

Several Mediators can and will coexist in the system. We assume that the communication between the PDS and the Mediator is secure in order to avoid ciphering invocations from the PDS to the Mediator. Typically, a Mediator will be instantiated in the machine that hosts the PDS although this is not strictly required (if a part of the network can be assumed to be secure). It is also possible to instantiate a different Mediator for each

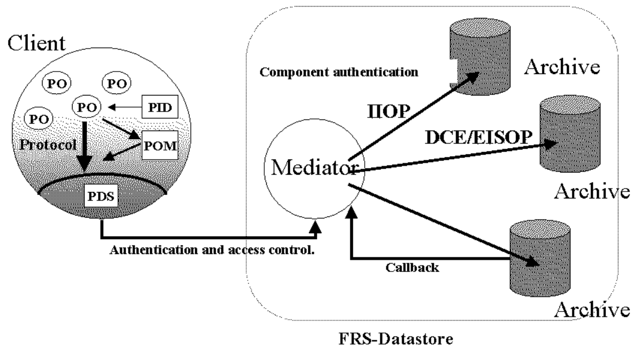

Figure 2: FRS-Datastore

user process or even for each persistent object. This is a configuration level policy that is outside the scope of this paper. For clarity, in the following text we just discuss the operation for a single persistent object bound to a given mediator.

In order to implement a store operation, the data is provided by the Persistent Data Service to the Mediator. The Mediator performs the fragmentation and establishes the scattering map, i.e., it selects which archives will store each fragment. Individual fragments are then sent randomly from the Mediator to the Fragment Stores in background. The fragmentation key is securely store in an Security Server.

In order to perform a restore operation, the Mediator recovers the fragmentation key from the Security Server and rebuilds the scattering map. It then issues "request for fragments" to the relevant Archives in order to obtain one copy of each fragment. Fragments are sent to the Mediator by call-backs. For additional security, the dissemination can also be scattered in the time domain (i.e., each Archive delays the call-back by a random time). The Mediator waits until all call-backs are received and reconstructs the object state. If one or more fragments are detected to be corrupted (or if one of the Archive sites fails during the process), the Mediator will issue additional "request for fragments" in order to obtain additional copies of the damaged fragments.

A fundamental aspect of our design is that the Mediator and the Archives interact through standard CORBA invocations. Thus, the FRS-Datastore is independent of ORB-specific functionalities. For increased intrusion tolerance, it is possible to configure the system such that the Mediator uses a different inter-ORB protocol to communicate with each individual Archive as illustrated in Figure 2 (this would require additional complexity for successful communication eavesdropping).

4 Interaction with the Persistence Service

The FRS-Datastore, as the name implies, is implemented at the datastore level of the CORBA Persistent Object Service Specification. In its simplest form, the datastore will offer an interface to save the state of the object in bulk (i.e., the complete raw state in a single store/restore invocation).

For performance reasons, it would be interesting to be able to update just a portion of the persistent state without performing a complete fragmentation and scattering operation. Since the object is paged, it should be possible to reconstruct and restore just the affected pages. However, since fragment names are obtained using a one-way function from the object identifier, and there is no explicit "fragment index" stored in the system, it is not obvious how such operation can be implemented efficiently without changing the naming algorithm (see also section 6).

5 Interaction with the Security service

Our goal is to use the mechanisms defined in the standard CORBA Security Service Specification to perform identification, authentication, authorization, access control and other security functions. The security service is responsible for authenticating the users and to store authorization information (service and file access rights).

Access control policies can be enforced to prevent unauthorized access to the FRS-Datastore. Access control policies can be enforced at different levels: principals not allowed to use the service at all, would be prevented from accessing a Mediator; the Mediator must check if an authenticated user is authorized to perform the requested operation (read/write/delete); additionally, each Archive site can impose site-specific restrictions on which principals are allowed to store fragments in the archive (we will return to this issue later). Privilege delegation is used to propagate the user rights through the object chain (PDS, Mediator, Archives and Security Server).

In terms of communication security the FRS-Datastore requires authentication functions (to mutually authenticate Mediators and Archives), integrity protection (this simplifies the design) but it does not requires confidentiality protection: this last issue is covered by the ciphering and scattering operation inherent to the FRS approach.

6 Open issues

In the Delta-4 architecture, each fragment is multicast to all archives which decide whether to store the fragment or not (reliable multicast was supported by the run-time system). During the retrieve operation, the list of fragments is broadcasted and each archive sends the fragments it owns. This implementation also assumes that the archives are either operational or failed. Basically, the implementation described in [1] heavily dependent on the availability of efficient reliable multicast primitives [5].

In our design, we are targeting a more open environment, where all interactions are based on standard point-to-point invocations. Also, we would like to make the service tolerant to network-partitions and to temporary disconnections of some of the Archives. We see two possible alternatives to extend this work into such direction, that we would eventually be pleased to discuss during the workshop.

The first alternative, that we call static scattering, consists in having the Mediators to make the scattering plan based on a static, pre-defined set of Archive sites, regardless of the availability at run-time. Temporary disconnections would be treated as failures for the matter of storing/retrieving fragments. The scattering replication degree would be configured such that the system, in run-time, has a high probability of storing/accessing at least one copy of each fragment.

The second alternative, that we call dynamic scattering, consists in having the Mediators to plan the scattering map according to the available Archives at the time the store operation is performed. This would also make easier to implement access control policies to each individual Archive site: before establishing the scattering map, the Mediator would inquire the Archives about their availability/authorization to store information on behalf of the principal. However, dynamic scattering requires the scattering map (or the list of Archives used) to be securely stored in the Security Service, along with the fragmentation key (such that the map can be later recovered for data retrieval).

Taking into account temporary disconnections introduces also another level of complexity. Since the fragment names should not disclosure any information about the data, there is no way to distinguish different "versions" of the same fragment. The only way to make such distinction is to create "new" fragments and to make the version number (or timestamp) a parameter of the one-way function used to name such fragments. Thus, some sort of "garbage collection" mechanism needs to be implemented to delete obsolete fragments from the Archives (note that not all the Archives that store an old "version" may be reachable when the new "version" is created). This same feature makes also difficult to perform updates to a small portion of the persistent state of an object without changing the whole persistent state.

7 Conclusions and future work

We have presented the design of a highly secure and fault-tolerant distributed datastore based on the CORBA model. The FRS-datastore has many configuration parameters that impact the quality-of-service in the security, fault-tolerance and performance domains (page size, ciphering method, replication level, scattering in the time domain, etc). We intend to gather experimental data from the prototype that we are currently building to get a better understanding of the tradeoffs in this framework.

References

[1] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed systems. In *IEEE Symposium on Research in Security and Privacy*, pages 110–121, Oakland (CA), USA, 1991.

[2] OMG. The Common Object Request Broker: Architecture and Specification, 1997.

[3] OMG. CORBAservices: Common Object Services Specification, 1997.

[4] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing.* ESPRIT Research Reports. Springer Verlag, November 1991.

[5] L. Rodrigues and P. Veríssimo. xAMp: a Multi-primitive Group Communications Service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 112–121, Houston, Texas, October 1992. IEEE.