The Middleware Architecture of MAFTIA: A Blueprint

Paulo Veríssimo
pjv@di.fc.ul.pt
FC/UL*

Nuno Ferreira Neves
nuno@di.fc.ul.pt
FC/UL

Miguel Correia
mpc@di.fc.ul.pt
FC/UL

Abstract

In this paper, we present the middleware architecture of MAFTIA, an ESPRIT project aiming at developing an open architecture for transactional operations on the Internet. The former is a modular and scalable cryptographic group-oriented middleware suite, suitable for supporting reliable multi-party interactions under partial synchrony models, and subject to malicious as well as accidental faults.

1 Introduction

In this paper, we present the middleware architecture of MAFTIA, a project aiming at developing an open architecture for transactional operations on the Internet. MAFTIA exploits common approaches to fault tolerance, of both accidental and malicious faults. The middleware platform is a distributed, modular and scalable cryptographic group-oriented suite, suitable for supporting reliable multi-party interactions under partial synchrony models, subject to malicious as well as accidental faults. A combination of intrusion prevention and tolerance measures is sought, under a hybrid failure model that identifies at least three classes of relevant faults: vulnerabilities, attacks, and intrusions. An extended version of this paper can be found in [9].

2 Failure Model

A crucial aspect of any architecture is the failure model upon which the system architecture is conceived, and component interactions are defined. The failure model conditions the correctness analysis, both in the value and time domains, and dictates crucial aspects of system configuration, such as the placement and choice of components, level of redundancy, types of algorithms, and so forth. There are essentially two different kinds of failure model: controlled failure assumptions; and arbitrary failure assumptions.

Failure Assumptions

Controlled failure assumptions specify qualitative and quantitative bounds on component failures. For example, the failure assumptions may specify that components only have timing failures, and that no more than f components fail during an interval of reference. Alternatively, they can admit value failures, but not allow components to spontaneously generate or forge messages, nor impersonate, collude with, or send conflicting information to other components. This approach is extremely realistic, since it represents very well how common systems work under the presence of accidental faults, failing in a benign manner most of the time. It can be extrapolated to malicious faults, by assuming that they are qualitatively and quantitatively limited. However, it is traditionally difficult to model the behaviour of a hacker, so we have a problem of coverage that does not recommend this approach unless a solution can be found.

Arbitrary failure assumptions specify no qualitative or quantitative bounds on component failures. Obviously, this should be understood in the context of a universe of ”possible” failures of the concerned operation mode of the component. For example, the possible failure modes of interaction, between components of a distributed system are limited to combinations of timeliness, form, meaning, and target of those interactions (let us call them messages). In this context, an arbitrary failure means the capability of generating a message at any time, with whatever syntax and semantics (form and meaning), and sending it to anywhere in the system.

Hybrid assumptions combining both kinds of failure assumptions would be desirable. Generally, they consist of allocating different assumptions to different subsets or components of the system, and have been used in a number of systems and protocols. Hybrid models allow stronger assumptions to be made about parts of the system that can

[*] Faculdade de Ciências da Universidade de Lisboa. Bloco C5, Campo Grande, 1749-016 Lisboa - Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EC, through project IST-1999-11583 (MAFTIA), by the FCT, through the Large-Scale Informatic Systems Laboratory (LASIGE) and by the project Praxis/33996/99 (DEFEATS).

justifiably be assumed to exhibit fail-controlled behaviour, whilst other parts of the system are still allowed an arbitrary behaviour. This is advantageous in modular and distributed system architectures such as MAFTIA. However, this is only feasible when the model is well-founded, that is, the behaviour of every single subset of the system can be modelled and/or enforced with high coverage, and this brings us back, at least for parts of the system, to the problem identified for controlled failure assumptions.

Composite Failure Model

The problems identified in our discussion of failure assumptions point to the need for the MAFTIA failure model to have characteristics enabling the definition of intermediate, hybrid assumptions, with adequate coverage. A first step in this direction is the definition of a composite failure model specifically aimed at representing the failures that may result from several classes of malicious faults. A second step is the definition of a set of techniques that act at different points within this composite failure model and which, combined in several ways, yield dependability vis-à-vis particular classes of faults. We are going to base our reasoning on two guiding principles:

- the sequence: attack + vulnerability → intrusion → failure

- the recursive use of fault tolerance and fault prevention

Concerning the mechanisms of failure, Figure 1 represents the fundamental sequence: *attack* + *vulnerability* → *intrusion* → *failure*. It distinguishes between several kinds of faults capable of contributing to a security failure. Vulnerabilities are the primordial faults existing inside the components, essentially design or configuration faults (e.g., coding faults allowing program stack overflow, files with root setuid in UNIX, naïve passwords, unprotected TCP/IP ports). Attacks are malicious interaction faults that attempt to activate one or more of those vulnerabilities (e.g., port scans, email viruses, malicious Java applets or ActiveX controls). An attack that successfully activates a vulnerability causes an intrusion. This further step towards failure is normally characterised by an erroneous state in the system that may take several forms (e.g., an unauthorised privileged account with telnet access, a system file with undue access permissions to the hacker). Such erroneous states can be unveiled by intrusion detection, as we will see ahead, but if nothing is done to process the errors resulting from the intrusion, failure of one or more security properties will occur.

The composite model embraced in MAFTIA allows the combined introduction of several techniques. Note that two causes concur to create an intrusion, as shown in Figure 1: vulnerabilities and attacks.

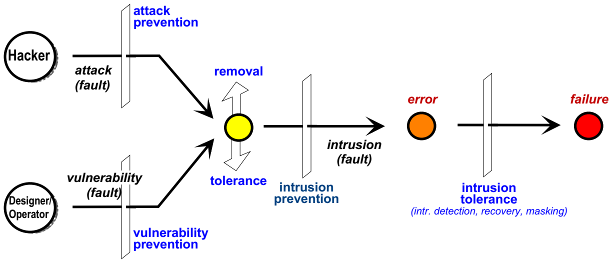

Figure 1. The Composite Failure Model of MAFTIA

To begin with, we can prevent some attacks from occurring, thereby reducing the level of threat imposed on the system. Attack prevention can be performed, for example, by selectively filtering access to parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, it cannot be attacked from there). However, it is impossible to prevent all attacks (e.g., some components have to be placed outside the firewall in a Demilitarised Zone), and in consequence, other measures must be taken.

On the vulnerability side, vulnerability prevention helps to reduce the degree of vulnerability by construction. However, many systems are assembled from COTS components that contain known vulnerabilities. When it is not possible to prevent the attack(s) that would activate these vulnerabilities, a first step would be to attempt vulnerability removal. Sometimes this is done at the cost of eliminating the system functions that contain the vulnerabilities.

The various combinations of techniques discussed above provide a range of alternatives for achieving intrusion prevention (see Figure 1), i.e. attempting to avoid the occurrence of intrusions. Whilst this is a valid and widely used approach, its absolute success cannot be guaranteed in all situations, and for all systems. The reason is obvious: it may not be possible to handle all attacks, either because not all attacks are known or new ones may appear, or because not all attacks can be guaranteed to be detected or masked. Similar reasoning applies to vulnerabilities. In consequence, some attacks will succeed in producing intrusions, requiring forms of intrusion tolerance, as shown in the right part of Figure 1, in order to prevent system failure. Again, these can assume several forms: detection (e.g., of intruded account activity, of Trojan horse activity); recovery (e.g., interception and neutralisation of intruder activity); or masking (e.g., voting between several components, including a minority of intruded ones) [4].

The above discussion has laid the foundations for achieving our objective: a well-founded hybrid failure model, that is, one where different components have different faulty behaviours. Consider a component for which a given controlled failure assumption was made. How can we achieve coverage of such an assumption, given the unpredictability of attacks and the elusiveness of vulnerabilities? The key is in a recursive use of fault tolerance and fault prevention. Think of the component as a system: it can be constructed through the combined use of removal of internal vulnerabilities, prevention of some attacks, and implementation of intrusion tolerance mechanisms internal to the component, in order to prevent the component from exhibiting failures.

Looked upon from the outside now, at the next higher level of abstraction, the level of the outer system, the would-be component failures we prevented restrict the system faults the component can produce. In fact we have performed fault prevention, that is, we have a component with a controlled behaviour vis-à-vis malicious faults. This principle:

- establishes a divide-and-conquer strategy for building modular fault-tolerant systems;
- can be applied in different ways to any component;
- can be applied recursively at as many levels of abstraction as are found to be useful.

Components exhibit a coverage that is justifiably given by the techniques used in their implementation, and can subsequently be used in the construction of fault-tolerant protocols under the hybrid failure model.

3 Synchrony Model

Research in distributed systems algorithms has traditionally been based on one of two canonical models: fully asynchronous and fully synchronous models [8]. Asynchronous models do not allow timeliness specifications. They are time-free, that is, they are characterised by an absolute independence of time.

Asynchronous models obviously resist timing attacks, i.e. attacks on the timing assumptions of the model, which are non-existent in this case. However, because of their time-free nature, asynchronous models cannot solve timed problems. However, timeliness is part of the required functionality of interactive applications, such as on-line operations on the stock market, multimedia, air traffic control.

Synchronous models allow timeliness specifications. In this type of model, it is possible to solve all hard problems (e.g., consensus, atomic broadcast, clock synchronisation) [3]. Synchronous models are characterised by having known bounds for processing and message delivery delays, and for the rate of drift and difference among local clocks.

However, synchronous models are fragile in terms of the coverage of timeliness assumptions such as: positioning of events in the timeline and determining execution durations. It is easy to see that synchronous models are susceptible to timing attacks, since they make strong assumptions about things happening on time.

Partial Synchrony

The introductory words above explain why synchronism is more than a mere circumstantial attribute in distributed systems subjected to malicious faults: *absence of time is detrimental to quality of service; presence of time increases vulnerability*. Intermediate timed partially synchronous models have deservedly received great attention recently. They provide better results, essentially for three reasons: (i) they allow timeliness specifications; (ii) they admit failure of those specifications; (iii) they provide timing failure detection. We are particularly interested in a model based on the existence of a timely computing base, which is both a timely execution assistant and a timing failure detection oracle that ensures time-domain correctness of applications in environments of uncertain synchronism [7].

This research has focused on benign (non-arbitrary, non-malicious) failure models. However, the architectural characteristics of the timely computing base enable its extension in order to be resilient to value- as well as time-domain failures. We call such an extended model, whose development we pursue in the MAFTIA project, a Trusted Timely Computing Base, or TTCB. In one sense, a TTCB has similar design principles to the very well known paradigm in security of a Trusted Computing Base (TCB) [1]. However, the objectives are drastically different. A TCB aims at fault prevention and ensures that the whole application state and resources are tamper-proof. It is based on logical correctness and makes no attempt to reason in terms of time. In contrast, a TTCB aims at fault tolerance: application components can be tampered with, but the whole application should not fail. In other words, a TTCB is an architectural artefact supporting the construction and trusted execution of fault-tolerant protocols and applications running under a partially synchronous model.

The architecture of a system with a TTCB is suggested by Figure 2. The first relevant aspect is that the heterogeneity of system properties is incorporated into the system architecture. There is a generic or payload system, over a global network or payload channel. This prefigures what is normally ”the system” in homogeneous architectures, that is, the place where the protocols and applications run. The latter can have any degree of synchronism, and be subjected to arbitrary attacks. Additionally, there is a control part, made of local TTCB modules, interconnected by some form of medium, the control channel. We will refer to this set up as the distributed TTCB, or simply TTCB when there is no ambiguity. The second relevant aspect of the TTCB is that its well-defined properties are preserved by construction, regardless of the properties of applications running with its assistance: it is synchronous, and it is trusted to execute as specified, being resilient to intrusions.

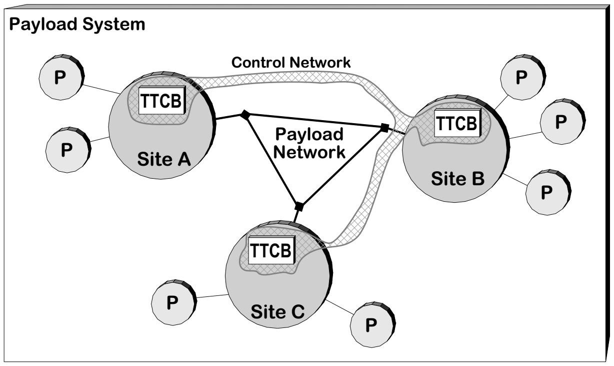

Figure 2. Trusted Timely Computing Base Model

Unlike the classic TCB, the TTCB can be a fairly modest and simple component of the system, used as an assistant for parts of the execution of the payload protocols and applications. Moreover, depending on the type of application, it is not necessary that all sites have a local TTCB. Consider the development of a fault-tolerant TTP (Trusted Third Party) based on a group of replicas that collectively ensure the correct behaviour of the TTP service vis-à-vis malicious faults. One possibility is for the replica group activity to be based on algorithms that support an arbitrary failure assumptions model (e.g., asynchronous randomised Byzantine Agreement), with the corresponding penalty in performance and lack of timeliness. Alternatively, the replica group management may rely on simpler algorithms that are at least partially executed in a synchronous subsystem with a benign (intrusion-free) failure model. Running these parts of the algorithm on a distributed TTCB substantiates the coverage of these assumptions.

4 Topological Model

Previous work on large-scale open distributed systems has shown the value of topology awareness in the construction of efficient protocols, from both functionality and performance viewpoints [6]. The principle is explained very simply: (i) the topology of the system is set up in ways that may enhance its properties; (ii) protocols and mechanisms in general are designed in order to recognise system topology and take advantage from it.

We intend to extrapolate the virtues of topology awareness to security in the MAFTIA architecture, through a few principles for introducing topological constructs that facilitate the combined implementation of malicious fault tolerance and fault prevention. The first principle is to use topology to facilitate separation of concerns: the site-participant separation in the internal structure of system hosts separates communication from processing; and the WAN-of-LANs duality at network level separates communication amongst local aggregates of sites, which we call *facilities*, from long-haul communication amongst facilities. The second principle is to use topology to construct clustering in a natural way. Two points of clustering seem natural in the MAFTIA large-scale architecture: sites and facilities. These principles are illustrated in Figure 3.

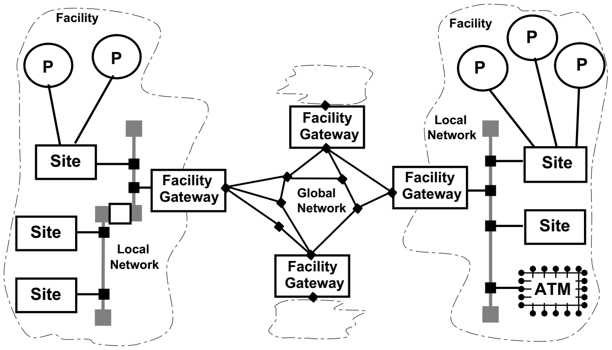

Figure 3. Two-tier WAN-of-LANs

Sites are hidden behind a single entry-point, a Facility Gateway, a logical gateway that represents the local network members, for the global network.

From a security viewpoint, participant-site clustering allows investing in the implementation of fault-tolerant and fault-preventive mechanisms at node level to collectively serve the applications residing in the node. On the other hand, the opportunities offered by site-facility clustering with regard to security are manifold: firewalling at the Facility Gateway; establishing inter-facility secure tunnels ending in the facility agents; inspecting incoming and outgoing traffic for attack and intrusion detection; ingress and egress traffic filtering; internal topology hiding through network address translation, etc.

System Components

The architecture of a MAFTIA node is represented in Figure 4, in which the local topology and the dependence relations between modules are depicted by the orientation of the (”depends-on”) arrows. In Figure 4 the set of layers is divided into site and participant parts. The site part has access to and depends on a physical networking infrastructure, not represented for simplicity. The participant part offers support to local participants engaging in distributed computations. The lowest layer is the Multipoint Network module, MN, created over the physical infrastructure. Its main properties are the provision of multipoint addressing and a moderate best-effort error recovery ability, both depending on topology and site liveness information.

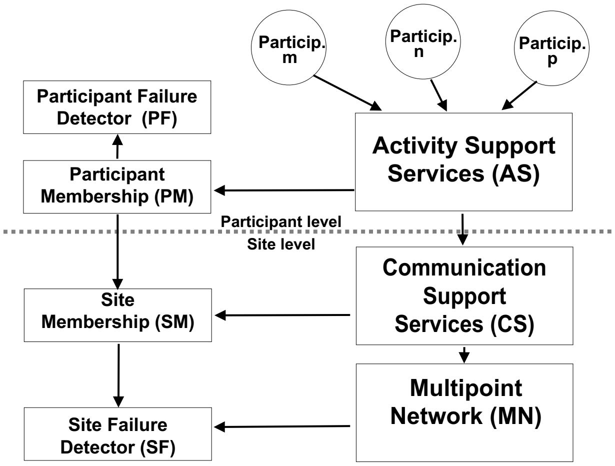
Particip. m
Particip. n
Particip. p
Participant Failure Detector (PF)
Participant Membership (PM)
Activity Support Services (AS)
Participant level
Site level
Communication Support Services (CS)
Site Membership (SM)
Multipoint Network (MN)
Site Failure Detector (SF)

Figure 4. Architecture of a MAFTIA Node

In the site part, the Site Failure Detector module, SF, is in charge of assessing the connectivity and correctness of sites, and the MN module depends on this information. The SF module depends on the network to perform its job, and thus is not completely reliable, due to the uncertain synchrony and susceptibility to attacks of at least parts of the network. The universe of sites being monitored can be parameterised, for example: all sites inside a facility, all sites having to do with ongoing computations at this site, all facility agents, etc. The Site Membership module, SM, depends on information given by the SF module. It creates and modifies the membership (registered members) and the view (currently active, or non-failed, or trusted members) of sets of sites, which we call site-groups. The Communication Support Services module, CS, implements basic cryptographic primitives (e.g., secure channels and envelopes), Byzantine agreement, group communication with several reliability and ordering guarantees, clock synchronisation, and other core services. The CS module depends on information given by the SM module about the composition of the groups, and on the MN module to access the network.

In the participant part, the Participant Failure Detector module, PF, assesses the liveness and correctness of all local participants, based on local information provided by sensors in the operating system support. The Participant Membership module, PM, performs similar operations as the SM, on the membership and view of participant groups. The PM module monitors all groups with local members, depending on information propagated by the SM and by the PF modules, and operating cooperatively with the corresponding modules in the concerned remote sites. The Activity Support Services module, AS, implements building blocks that assist participant activity, such as replication management (e.g., state machine, voting), leader election, transactional management, key management, and so forth.

Implementation

A prototype of the MAFTIA architecture is currently being developed with the collaboration of several partners. The middleware, in particular, has as major contributors IBM Zurich with the definition of some CS protocols [2], University of Newcastle upon Tyne who are extending their work on long running transactions [10], and University of Lisboa with the development of a secure group-communication suite on the partially-synchronous model [7]. Partners from DERA and University of Saarlandes are formally verifying and assessing some of the protocols that are being produced [5].

References

[1] M. Abrams, S. Jajodia, and H. Podell, editors. *Information Security*. IEEE CS Press, 1995.

[2] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. In *Proc. of the 19th Symposium on Principles of Distributed Computing*, July 2000.

[3] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996.

[4] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed systems. In *Proc. of the IEEE Symp. on Security and Privacy*, pages 110–121, May 1991.

[5] B. Pfitzmann, and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. of the Conf. on Computer and Communications Security*, Nov. 2000.

[6] L. Rodrigues and P. Veríssimo. Topology-aware algorithms for large-scale communication. In S. Krakowiak and S. Shrivastava, editors, *Advances in Distributed Systems*, LNCS 1752, chapter 6, pages 127–156. Springer Verlag, 2000.

[7] P. Veríssimo, A. Casimiro, and C. Fetzer. The timely computing base: Timely actions in the presence of uncertain timeliness. In *Proc. of the Int'l Conf. on Dependable Systems and Networks*, June 2000.

[8] P. Veríssimo, and M. Raynal. Time, clocks and temporal order. In S. Krakowiak and S. Shrivastava, editors, *Recent Advances in Distributed Systems*, volume 1752 of *LNCS*, chapter 1. Springer-Verlag, 2000.

[9] P. Veríssimo, N. F. Neves, and M. Correia. The Middleware architecture of MAFTIA: A blueprint. DI/FCUL TR 99-6, Department of Computer Science, University of Lisboa, Sept. 2000.

[10] J. Xu, B. Randell, A. Romanovsky, A. Stroud, R. Zorzo, E. Canver, and F. von Henke. Rigorous development of a safety-critical system based on coordinated atomic actions. In *Proc. of the 29th Symposium on Fault-Tolerant Computing Systems*, pages 68–75, June 1999.