SAFETY AND SECURITY ARCHITECTURES FOR AVIONICS*

Youssef Laarouchi, Yves Deswarte, David Powell, Jean Arlat
LAAS-CNRS, Université de Toulouse, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
youssef.laarouchi@laas.fr, yves.deswarte@laas.fr, david.powell@laas.fr, jean.arlat@laas.fr

Keywords: Critical systems, integrity levels, virtualization, information flow validation.

Abstract: In computer systems, commercial off-the-shelf (COTS) components offer extended functionalities for a reasonable cost, and consequently have an important economic advantage. However, such components are hard to integrate into critical systems because of the integrity requirements placed on such systems. To alleviate this problem, we consider the use of Totel's integrity model (Totel et al., 1998), a model for managing multiple levels of integrity and allowing the use of fault tolerance techniques to validate information flow from low integrity components to high integrity ones. We propose the use of virtualization as a means to diversify COTS components running on the same physical machine and to control information flow on this machine.

1 INTRODUCTION

In most critical application domains, certification authorities mandate strictly constrained processes for the development and validation of critical systems, which significantly increases the production cost of such systems. At the same time, commercial off-the-shelf (COTS) components offer attractive prospects for low-cost extended functionalities. However, COTS components are generally less dependable than well-validated critical components, which makes their integration into a critical system problematic. One solution is to completely isolate critical components from less critical ones to ensure that they cannot be affected by errors propagated from faulty or corrupt COTS components. However, this solution is much too restrictive since it does not offer any possibility for interaction between components having different integrity levels.

Another more flexible solution is to ensure unidirectional information flow from high integrity levels to lower levels. In this case, critical software can control or send data to COTS components, but cannot receive any information (including acknowledgements) from them. This form of interaction can be likened to diodes used in electronic circuits. But this solution is still too restrictive: some critical control applications need to have a global view of the system, including low criticality applications, in order to make consistent decisions. This cannot be achieved without some upwards flow of information.

Several solutions have been proposed to allow communications between applications having heterogeneous integrity levels: Biba's model (Biba, 1977), Clark and Wilson's model (Clark and Wilson, 1987), Integrated Modular Avionics (IMA) (Rushby, 2000), the causal dependencies principle (d'Ausbourg, 1994). In this paper, we consider Totel's model (Totel et al., 1998), which, by checking all the information flows existing in a system, allows bidirectional communications between software components at different integrity levels. In the ArSec (*Architectures de Sécurités*) project, we are studying the application of this model in the avionics context so that information coming from low integrity level software can be used by critical avionic components, without negatively impacting the overall system safety. Such flows have

*This study is part of the ArSec Project, a cooperation between LAAS-CNRS and AIRBUS France under AIRSYS convention. This study is also partially supported by the European Network of Excellence ReSIST (Resilience for Survivability in IST), funded by the European Commission Framework Program 7, Contract Number 026764, <http://www2.laas.fr/RESIST/index.html>.

been identified between heterogeneous-integrity-level components embedded in an aircraft. Other new interactions between on-board and ground entities have also to be fully checked.

In this paper, we first present Totel's model (Section 2) with its interpretation and adaptation in the ArSec project (Section 3). Then we describe in Section 4 the use of virtualization techniques to implement Totel's model. Finally in Section 5 we provide concluding remarks and present our future work.

2 TOTEL'S MODEL

2.1 Model presentation

This model was introduced by Eric Totel in his Ph.D. thesis work. Figure 1 presents the global software architecture that has to be deployed to allow bidirectional communications between objects having different integrity levels. Objects may be software components or complete hardware/software sub-systems. The direction of arrows corresponds to the direction of logical information flow.

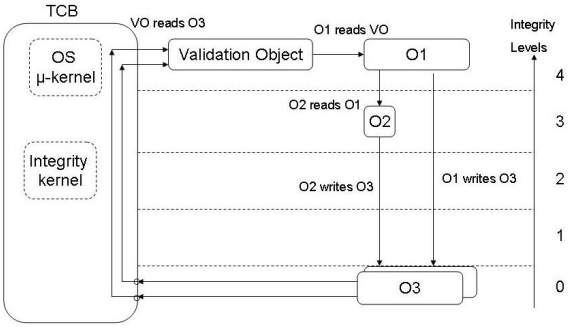

Figure 1: Totel's model for multilevel integrity components

As mentioned, downward communication does not adversely affect operation of critical tasks. Thus, downward communication does not need any specific checking and is allowed. However, direct upward communication is forbidden. The only way for a high-integrity-level object to receive information from a low-integrity-level one is to use a specific communication channel. This channel is ensured by a Trusted Computing Base (TCB) that offers a specific entry point for each upward communication flow. To avoid any violation of this principle, the TCB checks all communication in the system and implements the diode mechanism to prevent direct upward information flow. Only specific upward flows are allowed: towards *Validation Objects* (VOs).

In Figure 1, level 0 objects need to send information to object $O1$, which is the most critical one in the system. We have to ensure that information coming from the lower levels does not alter the behavior of $O1$. To do this, all information coming from lower levels has to be validated by a Validation Object (VO) associated with $O1$. VO is the only object in the model allowed to receive data from lower integrity levels. It must have the integrity level of the object for which it validates information. VO contains fault tolerance mechanisms to check the data flow. The implementation of VO is tightly linked to the type of information that it checks. For example, in Figure 1, VO can compare or vote on redundant input data provided by the two instances of object $O3$.

The TCB of (Totel et al., 1998) consists of the information checking mechanisms, a microkernel and an integrity kernel. The microkernel is used to schedule tasks being run by different objects and to completely isolate the address spaces of each object. The integrity kernel implements integrity checking mechanisms that ensure that information flows are in the correct direction and are authorized between specified entry and exit points of the TCB. The TCB must interact with the most critical objects of the system, so it has to be validated to the highest integrity level.

2.2 Limitations

The TCB represented in Figure 1 contains a microkernel because the presented model deals only with one physical machine. Consequently, we have to adapt this architecture to the distributed runtime environment relevant for the avionics application we are considering. As mentioned in section 2.1, the TCB has the integrity level of the most critical objects for which it checks information. Thus, in a multiple-integrity-level system, the TCB has to be developed and validated to the highest level, if the checked information traffic is towards that level. However, we have to take into consideration the fact that a high-integrity-level component has to be as simple as possible so that it it is feasible to validate it at this level. Therefore, the development complexity of each element of Totel's model has to be taken into account while implementing it. In the next section, we present an adaptation of Totel's model to a distributed context, such as the one in avionics. Development complexity and validation by certification authorities are especially accounted for.

3 DISTRIBUTED CONTEXT ADAPTATION

To adapt Totel's model to a distributed context, we have to study the characteristics of each entity introduced in the model, especially the trusted computing base and the validation objects. We first consider TCB properties and then the decomposition of the TCB in a distributed context.

3.1 TCB properties in Totel's model

In a single machine context, the TCB contains a microkernel and an integrity kernel (*cf.* Figure 1). We have studied the characteristics of these two kernels and established the environment-independent properties that the TCB has to implement. These properties are: non bypassability, spatial partitioning and temporal partitioning. The two types of partitioning correspond to the two interaction types between software components. Spatial interaction is concerned with data exchange between components (by direct modification of shared memory or by any other form of communication). Temporal interaction is concerned with timing effects between component executions, e.g., through contention for common resources.

3.1.1 Non bypassability

The non bypassability property ensures that all interactions between heterogeneous integrity levels are subject to checking by the TCB's integrity kernel. Downwards and same-level communications (i.e., from level i to level j where $i \geq j$) are authorized. Upwards communications from level i to level j ($i < j$) must go through the TCB's integrity kernel. This property ensures that the TCB has absolute control over all information flows and avoids any violation of communication policies between different integrity levels, as described in Section 2.

3.1.2 Spatial partitioning

The spatial partitioning property ensures separation between memory spaces (or any other communication channel) accessible by software components with different integrity levels. This aims to avoid component corruption through direct memory modification or through exchanging erroneous data that could provoke the failure of these components. Spatial partitioning between integrity levels forbids any form of communication between levels that has not been approved by the TCB.

3.1.3 Temporal partitioning

Temporal partitioning avoids any temporal interaction between components having different integrity levels. For example, a faulty or corrupt COTS application could monopolize or block the access to the CPU and prevent critical software from executing. In this case, the TCB controls access to the CPU by the COTS application so that critical applications using the same CPU can run without being disturbed. In a distributed context, a non-critical component must not be able to occupy more than a specified proportion of the bandwidth of any network shared with a critical application.

3.2 TCB decomposition

The TCB properties have to be satisfied in any execution environment, and especially in a distributed one. In that case, we propose to decompose the TCB (*cf.* Figure 1) into several separate TCBs, each one being responsible for upwards information flow from a level i to a more critical level j. So, going from level 0 to level 2 and from level 0 to level 3 in Figure 2 is respectively ensured by separate TCBs: $TCB1$ and $TCB2$. This aims to reduce TCB complexity since it has fewer flows to manage.

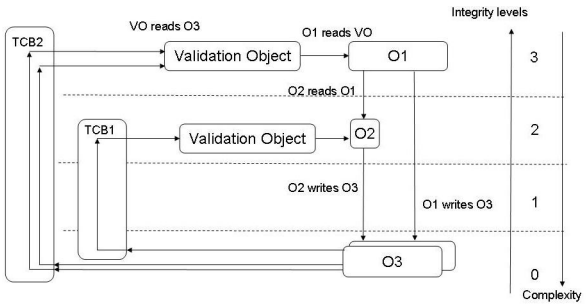

Figure 2: TCB decomposition in Totel's model

In Figure 2, $TCB2$ has to carry data from integrity level 0 to integrity level 3. In avionics, this corresponds for example to a transfer of information from a component at design assurance level (*DAL*) E to a component at DAL B of the DO-178B standard (RTCA, 1992). If the DAL E component is a COTS component, it may implement quite complex communication primitives, which would in turn require the development of a commensurately complex TCB. However, the more complex the TCB is, the more difficult will be its validation for it to be accepted by certification authorities. Thus we have to limit interaction between TCBs and low-integrity-level software. For this reason, we introduce a new

intermediary object category, called a *proxy*. Proxies have an intermediate integrity level, greater than the one below and less than the one above. In this way, information going from level i to a level k $(k > i+1)$ has to go through a proxy having level j, where $i < j < k$, as shown on Figure 3.

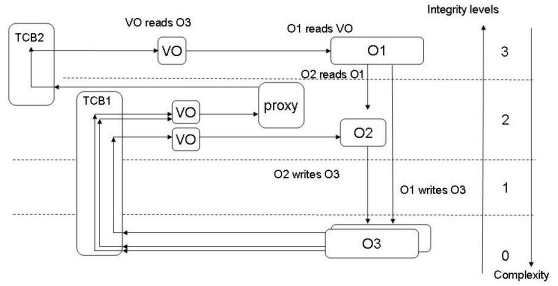

Figure 3: The use of proxies in Totel's model

The use of proxies reduces TCB complexity and facilitates its validation for acceptance by certification authorities. Indeed, the use of a level 2 proxy in Figure 3 would lead to a less complex $TCB2$ than in Figure 2, since it no longer needs to span as many integrity levels.

4 VIRTUALIZATION AND TOTEL'S MODEL

In Totel's model, validation objects are the only entities in the system allowed to receive data from low-integrity-level objects. VOs implement fault tolerance mechanisms (Avizienis et al., 2004) to increase the integrity level of received information. These mechanisms are essentially based on redundancy and diversification. The diversification can be topological (by using geographically-separated machines having the same configuration), functional (by using different designs aimed at meeting the same requirement), temporal (by executing the same software at different moments), physical (by using different hardware) and/or translational (by using different compilers).

In the ArSec project, some COTS components are located on the same physical machine and use the same hardware. Consequently, we can only implement functional, translational and temporal diversification. Temporal and translational diversification are not robust enough against malicious faults. Indeed, if a hacker succeeds in attacking software code, the temporal and translational diversification will likely reproduce the same behavior by all redundant components. Functional diversification is more adapted to a context of malicious attacks. The only way to support this type of diversification on the same machine is to use the virtualization technique.

Virtualization allows two or more operating systems to run simultaneously on the same machine. Using virtualization has the advantage that the virtual machine monitor (VMM) natively implements the required spatial and temporal isolation properties. It has also the advantage of supporting isolated operation of different operating systems so that attacks on one OS do not affect the others. Thus, on a single physical machine, the VMM instantiates different operating systems and schedules them to run in a non conflicting way. The VMM has control of all I/O and can easily implement mechanisms for integrity checking. Obviously, the VMM has to be integrated in the TCB and validated to a higher integrity level than the objects running on COTS OSes. Validation objects have also to be validated at the same level as the VMM, and could be integrated in the VMM.

Figure 4 describes the architecture of a single machine implementing Totel's model using the virtualization technique.

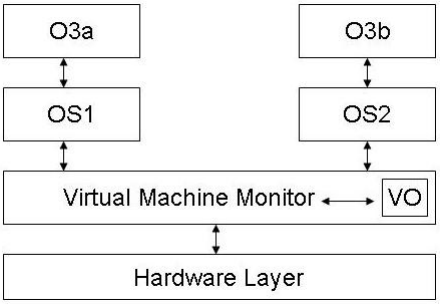

Figure 4: Implementing VO in a Virtual Machine Monitor

Different versions of $O3$ ($O3a, O3b$) can be instantiated on different OSes ($OS1$ and $OS2$). Their outputs are cross-checked by a validation object that compares the results and provides a trustworthy output. All communications between $O3$ versions and VO are checked by the VMM. The trustworthy output delivered by VO may be sent to a critical external object, e.g., an embedded airborne software component.

The integration of VO into the VMM leads to a more complex VMM. Alternatively, VO could be supported by a certified OS (running as a guest virtual machine), thereby reducing the required VMM validation effort. In Figure 5, $OS3$ has to be certified at least to the integrity level of the hosted VO. Since this OS controls VO inputs and outputs, it must be considered as a part of the TCB.

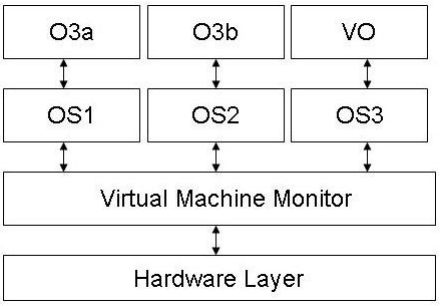

Figure 5: Implementing VO on a certified guest OS

5 CONCLUSIONS

We have presented the different research directions being studied in the ArSec project, especially a distributed software architecture for managing multiple levels of integrity and its implementation in the avionics context using virtualization.

Virtualization is the only way to manage diversified variants on one physical machine. This technique natively implements the isolation properties required of a TCB and facilitates implementation of functional diversification.

However, the use of one physical machine raises a design issue regarding the human-machine interface (HMI) in that a single human has to interact, in fault-tolerant way, through a single video display and keyboard, with multiple diversified software components. How to address this issue is the topic of ongoing work.

ACKNOWLEDGEMENTS

The authors of this paper thank M. Eric de Nadai, AIRBUS France, for his contributions in the present research work.

REFERENCES

Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33.

Biba, K. (1977). Integrity Considerations for Secure Computer Systems. *MITRE Co., technical report ESD-TR 76-372*.

Clark, D. and Wilson, D. (1987). A Comparaison of Commercial and Military Computer Security Policies. *in IEEE Symposium on Security and Privacy, Oakland, CA, pp. 184-194, IEEE Press*.

d'Ausbourg, B. (1994). Implementing Secure Dependancies Over a Network by Designing a Distributed Security Subsystem. *in European Symposium on Research in Computer Security (ESORICS'94), Brighton, UK, pp. 247-266, LNCS 875, Springer*.

RTCA (1992). Software Considerations in Airborne Systems and Equipment Certification. *Radio Technical Commission for Aeronautics (RTCA), European Organization for Civil Aviation Electronics (EUROCAE), DO178-B*.

Rushby, J. (2000). Partitionning in Avionics Architectures: Requirements, Mechanims, and Assurance. *NASA report, CR-1999-209347*.

Totel, E., Blanquart, J.-P., Deswarte, Y., and Powell, D. (1998). Supporting multiple levels of criticality. *in 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28), Munich, Germany, pp. 70-79, IEEE CS Press.*