# Patterns for the secure and reliable execution of processes

Eduardo B. Fernandez[1] and David laRed Martinez[2]

[1]Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL , USA

[2]Dpto. de Informática
Facultad de CienciasExactas
Universidad Nacional del Nordeste
Corrientes, Argentina

## Abstract

The controlled interaction of processes in a computing environment is fundamental for its security and reliability. Processes can be attacked by other processes or by external clients, errors in one process can propagate to others. We show here three patterns that can help provide a secure and reliable execution environment although they need to be complemented with other patterns. They include Protected Entry Points, which control the correct use of entry points according to their signatures (type and length of parameters); and Protection Rings, which control the calls between processes, enforcing constraints on entry points and signatures according to the level of trust in the processes. Finally, the Multilevel Secure Partitions (MSP) pattern, confines execution of a process to a system partition that has a specific confidentiality or integrity level.

## 1. Introduction

In a computer system processes typically collaborate to perform some activity or call each other to request services. Process invocations occur through procedure calls or remote procedure calls; these operations are supported at the kernel level through send/receive operations, which may be direct or indirect (using mailboxes) [Sil05]. The operation name used for invocation, plus the number, type, and length of the parameters in the call is called the procedure *signature*. The controlled interaction of processes in a computing environment is fundamental for its security and reliability. Processes can be attacked by other processes or by external clients, errors in one process can propagate to others. Executing processes in a computing system need to be protected from attacks from other processes. Many of those attacks come from the invocation of unprotected (no access control) or wrong entry points or using the wrong type or size of parameters in these calls. We present here two patterns to prevent these attacks: *Protected Entry Points* and *Protection Rings*. Protected Entry Points control the correct use of entry points according to their signatures. The entry points may apply additional checks, e.g. access control. Protection Rings control the calls between processes, enforcing constraints on entry points and signatures according to the level of trust in the processes.

Many security attacks propagate through weak parts of a system. After finding an entry point, the malicious software may access a directory or some other unit of the architecture, frequently escalating its power. We need to find ways to stop or obstruct this propagation. We present here a pattern that defines a partitioning of the system with this purpose. The *Multilevel Secure Partitions (MSP)* pattern is based on the properties of the multilevel access control model [Gol06]. It confines execution of a process in a system partition that has a specific confidentiality or integrity level. Access from this process to other partitions is performed through specific, protected entry points, restricted according to the rules of a multilevel security model.

Figure 1 shows how these patterns relate to each other. The Protected Entry Point pattern provides a mechanism for controlling calls between processes. The Protection Rings pattern and the Multiple Secure Partitions pattern restrict interprocess calls. The MSP pattern uses the multilevel model to control calls.

Section 2 discusses the Protected Entry Points pattern while Section 3 presents the Protection Rings pattern . Section 4 discusses the MSP pattern and we end with some conclusions and ideas for future work.

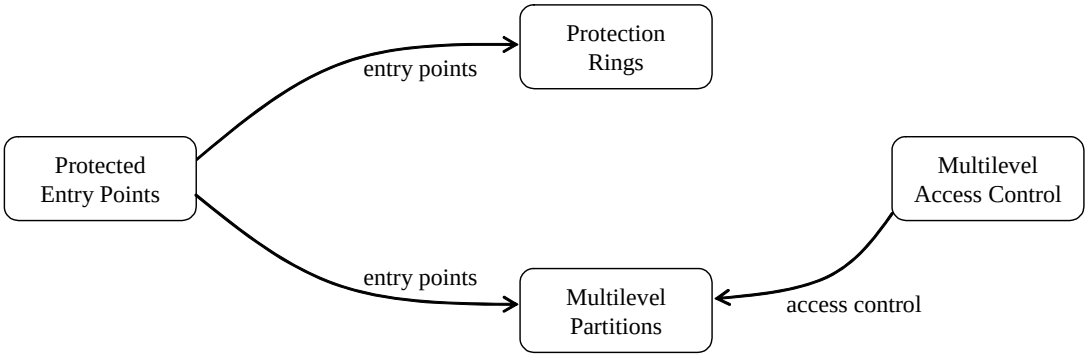


Figure 1. Pattern diagram of the patterns in this paper.

## 2. Protected Entry Points

This pattern forces a call from a process to another to go through only pre-specified entry points where the correctness of the call is checked and other access restrictions may be applied.

### 2.1 Example

ChronOS is a company building a new operating system, including a variety of plug-in services such as media players, browsers, and others. In their design, processes can call each other in unrestricted ways. This makes process calls fast, which results in general good performance and everybody is satisfied. However, when they test the system, an error anywhere produces problems because it propagates to other processes, corrupting their execution. Also, many security attacks are shown to be possible. It is clear that when their systems are in use they will acquire a bad fame and they will have problems selling it. We need to have a system which provides resilient service in the presence of errors and resistant to attacks.

### 2.2 Context

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

### 2.3 Problem

Process communication has an effect on security because if a process calls another in entry points without appropriate checks the calling process may read or modify data illegally, alter

the code of the executing process, or take over its privilege level. If the checks are applied in specific entry points, some languages, e.g. C or C++, let the user manipulate pointers to bypass those entry points. Process communication also has a strong effect on reliability because an error in a process may propagate to others and disrupt their execution.

The solution is affected by the following forces:

- Executing processes need to call each other to perform their functions. For example, in operating systems user processes need to call kernel processes to perform I/O, communications, and other system functions. In all environments, process may collaborate to solve a common problem and this collaboration requires communication. All this means that we cannot use process isolation to solve this problem.
- A call must go to a specified entry point or checks could be bypassed. Some languages let users alter entry point addresses so input checks can be bypassed.
- Typically, a process provides services to other processes but not all services are available to all processes. A call to a service not authorized to a process can be a security threat or allow error propagation.
- In a computing environment we have a variety of processes with different levels of trust. Some are processes which we normally trust, such as kernel processes, others may include operating system utilities, user processes, and processes of uncertain origin. Some of these processes may have errors or be malicious. All calls need to be checked.
- The number, type, and size of the passed parameters in a call can be used to attack a process, e.g. by producing a buffer overflow. Wrong parameters may produce or propagate an error.

### 2.4 Solution

Systems that use explicit message passing have the possibility of checking each message to see if it complies with system policies. For example, a security feature that can be applied when calling another process is *protected entry points*. A process calling another process can only enter this process at pre-designed entry points and only if the signature used is correct (name, number of parameters, type and size of parameters). This prevents bypassing entry checks and avoids attacks such aa buffer overflows.

#### *Structure*

Figure 2 shows a class diagram of the solution. **Calling** and **Called Processes** are roles of processes in general. When a Calling Process makes a request for a service to another process, the request is handled by an **Entry Point**. This entry point has a name and a list of parameters with predefined numbers, types, and size limits that can be used to check the correction of the call signature. It can optionally add access control checks by using a Reference Monitor pattern or other input data tests.

#### *Dynamics*

Figure 3 shows a process performing a service call. The call must use a proper signature, i.e., if the name of the service (opName) or the names of the parameters are incorrect, and the type or length of the parameters is not correct, it is rejected (this is checked by operation checkParmList).

### 2.5 Implementation

As mentioned earlier, kernels support calls as direct calls or through mailboxes. In the first case, the called process must check that the call is correct; in the second case, the mailbox must do the checking.

Entry points must be expressed as references as in Java, and not as pointers as in C or C++ (pointers allow arithmetic operations). In languages that use pointers, it is necessary to restrict their use in procedure calls, e.g. no pointer arithmetic.

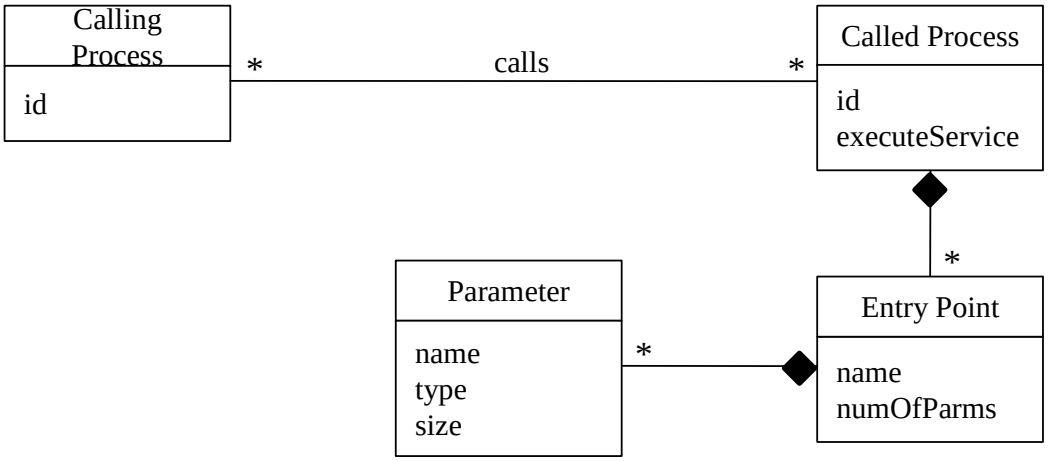


Figure 2. Class diagram of the Protected Entry Point pattern

**2.6 Example resolved**

If parameters of all calls are validated through Protected Entry Points, many security and reliability problems can be avoided. Additional checks, such as access control and data value checks can also be applied.

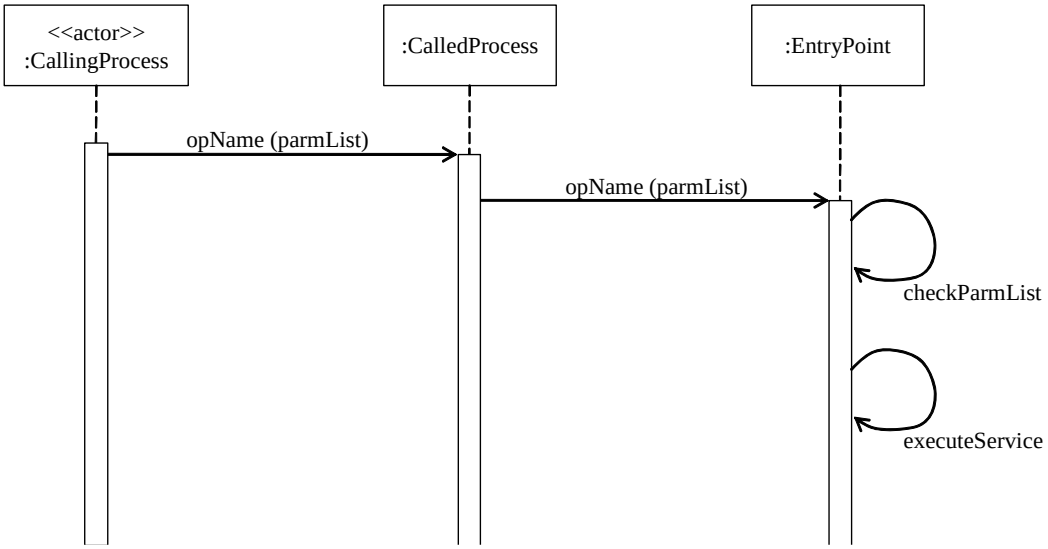


Figure 3. Sequence diagram for a process making a service call.

### 2.7 Known uses

- Multics
- Systems that use ring architectures, e.g. the Intel Series 86 and Pentium.
- Systems that use capabilities, e.g. IBM's S/6000.
- A specific use can be found in a patent for PC BIOS [Day91].

### 2.8 Consequences

This solution has the following advantages:

- If we can check all the calls of a process into another, we can check that the calls are for appropriate services and apply checks for security or reliability purposes.
- Checking the number, type, and length of the parameters passed in a call, can prevent a variety of attacks and stop the propagation of some errors.
- If we know the level of trust of processes we can adjust the number of checks; for example we apply more checks to suspicious processes.

### 2.9 Related patterns

- This pattern can be seen as a specific realization of an abstract principle: "Validate input parameters".
- Protection Rings, see Section 3.
- Multilevel Secure Partitions, see Section 4.
- Capabilities [Gol06].
- Access Control [Sch06] and Distributed Access Control [Del07]. These checks can be applied in specific entry points to control access to resources.

## 3. Protection Rings

Assign processes to a set of hierarchical rings that control how processes call other processes and how they access data. Crossing of rings is done through *gates* that check the rights of the crossing process. A process calling a process or accessing data in a higher ring must go through a gate.

### 3.1 Example

The ChronOS designers found that for applications that use programs with a variety of origins, there is a high overhead in applying elaborate checks to all of them. It would be more efficient to apply the checks selectively depending on how much we trust the programs making the calls but we usually don't know that at execution time. If we could have a way to classify processes according to trust, we could improve the application of checks. Also, we cannot rely on program features to enforce entering the right entry points because applications may come in a variety of languages, some of which may allow skipping entry points.

### 3.2 Context

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. Some processes may be malicious or contain errors that may affect process execution. This pattern applies only to centralized environments.

### 3.3 Problem

Defining a set of protected entry points is not enough if we cannot enforce their use. How can we prevent a process from calling another in an entry point which has no checks? We cannot

rely on language features unless we only use a restricted set of languages, not practical in general. If all processes are alike we also need to apply the same checks to all of them, which may be an overkill.

The solution is constrained by the following forces:

- We want to be able to enforce the application of Protected Entry Points, at least for some processes. In this way, requests from suspicious processes can be always controlled.
- We would like to separate processes according to their level of trust and check only calls from a low-level to a higher-level process. This can reduce considerable execution-time overhead.
- In each higher-level we want to check signature validity as well as access control or reliability tests. These actions should result in a more secure execution environment.

### 3.4 Solution

Define a set of hierarchical protection domains, called *protection rings* (typically 4 to 32) with different levels of trust. Processes are assigned to rings based on their level of trust. Ring crossing is performed through gates that enforce protected entry points: A process calling a higher-level process or accessing data at a higher level can only enter this process or data at pre-designed entry points with controlled parameters. Additional checks for security or reliability can be applied at the entry points.

***Structure***

Figure 4 shows a class diagram for this pattern. The **Calling Process** requests services from a **Called Process**. To do so, it must enter a **Call Gate**, that applies **Protected Entry Points**, that check the correct use of signatures. **Call Rules** define the requirements for interlevel calls. The **Calling Process** can access **Data** according to a set of **Data Access Rules**.

***Dynamics***

Figure 5 shows a sequence diagram for a call to a higher privilege ring. If the call fails an exception may be raised.

**Variants**

Rings don’t need to be strictly hierarchic, partial orders are possible and convenient for some applications. For example, a system including a secure database system could assign a level to this database equal but separated from system utilities; the highest level is for the kernel and the lowest level is for user programs. This was done in a design involving an IBM 370 [Fer78].

In some systems, e.g. the MV8000, rings are associated with memory locations.

Multics used the concept of call bracket. See Section 3.5.

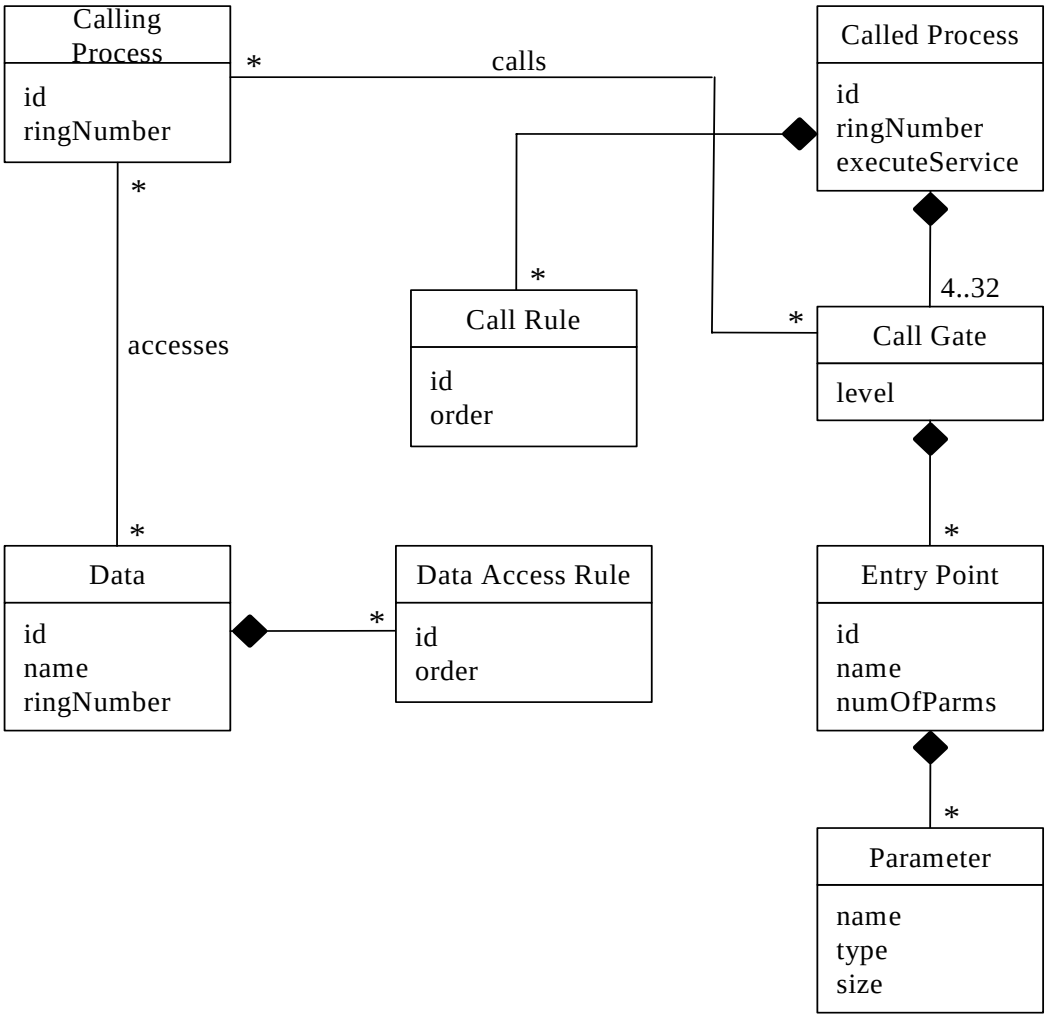


Figure 4. Class diagram for the Protection Ring pattern.

**3.5 Implementation**

The Call Rules and the Data Access Rules are usually implemented in the Call instruction microcode [int99]. Figure 6 shows a typical use of rings. Processes are assigned to rings based on their level of trust; for example, we could assign four rings in decreasing order of privilege and trust to: supervisor, utilities, trusted user programs, untrusted user programs.

The Program Status word of the process indicates its current ring and data descriptors also indicate their assigned rings. The values of the calling and called processes are compared to apply the transfer rules.

The Intel X86 architecture [int99] applies two rules:

- Calls are allowed only in a more privileged direction, with possible restriction of a minimum calling level.
- Data at level p can be accessed only by a program executing at a more privileged level (<= p).

Another possibility to improve security is to allow calls only within a range of rings; in other words jumping many rings is considered suspicious. Multics defined a *call bracket*, where calls are allowed only within rings in the bracket. More precisely, for a call from procedure i to a procedure with bracket (n1, n2, n3) the following rules apply: if n2<i<=n3 the call is allowed to specific entry points; if i>n3 the call is not allowed, if i < n1 any entry point is valid. This extension only makes sense for systems that have many rings.

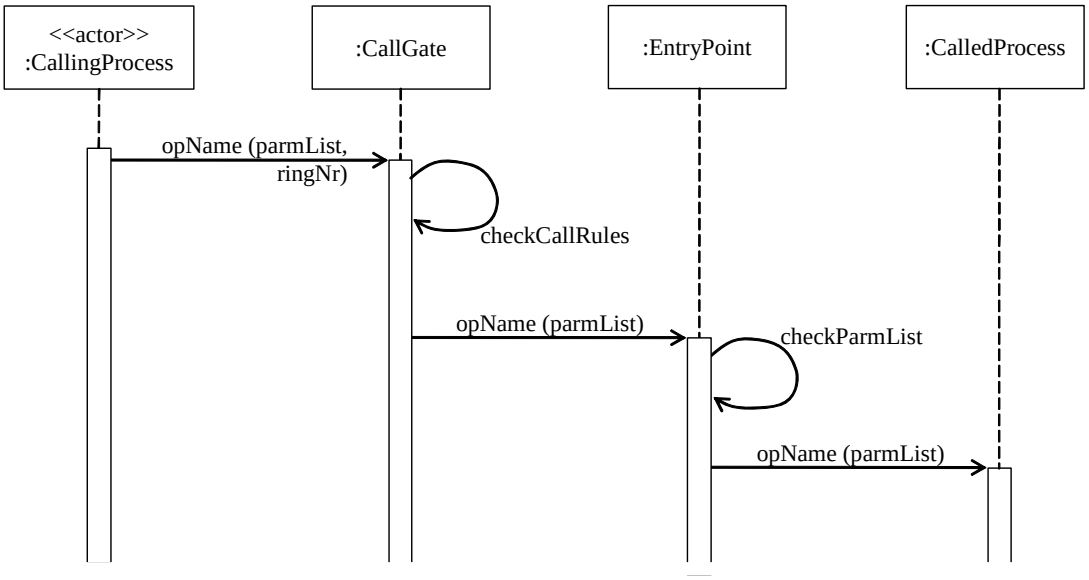


Figure 5. Sequence diagram for a successful call to a higher-privilege ring

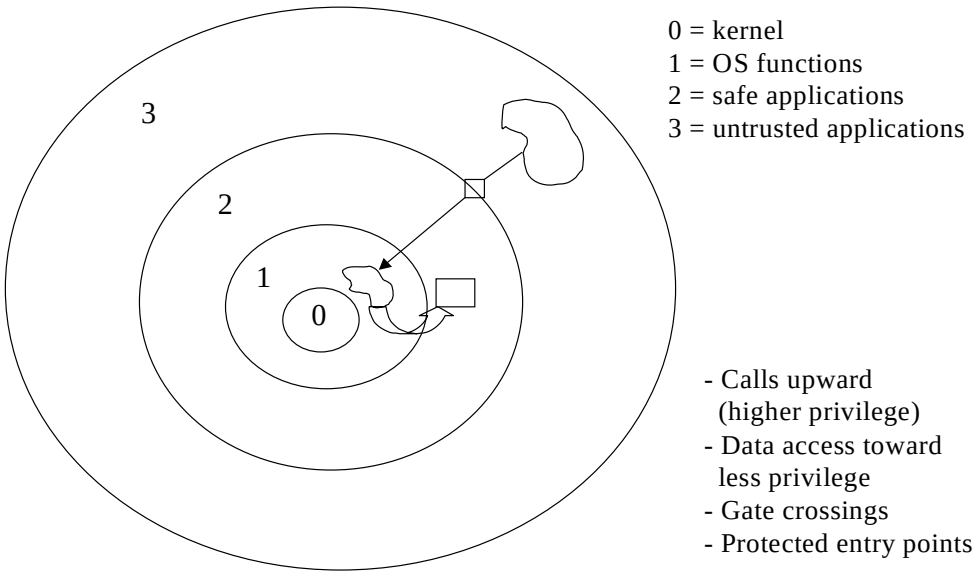

Figure 6. Assignment of protection rings

### 3.6 Example resolved

Now we can pre-assign processes to levels according to their trust. All calls to processes of higher privilege are checked. Processes of low trust get more checks.

### 3.7 Known uses

- Multics introduced this concept and used 32 rings as well as call brackets (see Section 3.5) [Gra68].
- The Intel Series X86 and Pentium [int99].
- MV8000 [mv, Wal81].
- Hitachi HITAC
- Other computers using this idea are the ICL 2900, VAX 11, and MARA, described in [Fro85], which also describes Multics and the Intel series.
- [Shi00] shows a use of rings to protect against malicious mobile code.
- An IBM S/370 was modified to have non-hierarchical rings [Fer78]
- Rings have been used for fault-tolerant applications [Oza88].

### 3.8 Consequences

This pattern has the following advantages:

- We can separate processes according to their level of trust.
- Level transfers happen only through gates where we can apply Protected Entry Points; that is, we have enforced protected entry points for upward calls.
- We can control procedure calls as well as data access across levels.

Possible disadvantages include:

- Crossing rings take time. Because of this delay some operating systems use fewer rings. For example, Windows uses 2 rings, IBM's OS/2 uses 3 rings [wik]. Using fewer rings improves performance at the expense of security.
- Without hardware support the crossing ring overhead is unacceptable, which means that this approach is only practical for operating systems and for centralized environments.

### 3.9 Related Patterns

A combination (process, domain) corresponds to a row of the Access Matrix [Sch06].

- Multilevel Secure Partitions (See Section 4). That pattern is an alternative for distributed environments, where processes are assigned levels based on multilevel security models [Gol06].
- Protected Entry Points (See Section 2).

## 4. The Multilevel Secure Partitions pattern

Confines execution of a process in a system partition which has been assigned to a specific confidentiality or integrity level. Access from this process to other partitions (processes or data) is restricted according to the rules of a multilevel security model, where processes have sensitivity levels.

### 4.1 Example

ChronOS is now building a web system. The system's Web, Application, and Database Servers are separated because it is clear that the Web Server has a higher exposure to attacks. If the Web server is compromised, ChronOS will be unable to provide some business services to its clients but at least the hacker is denied immediate access to Application and Database servers where the corporate data is stored. ChronOS wants to limit the damage from any one attack and prevent the attacker erasing their steps from the logs. The normal operation of the application requires processes to request and obtain services between the elements of the system.

**4.2 Context**

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

In multilevel models data and procedures are classified into sensitivity levels and users have access to them according to their clearances. These models have been formalized in three different ways [Gol06]:

- The Bell-La Padula model, intended to control leakage of information between levels.
- The Biba model, which controls data integrity.
- The Lattice model, which generalizes the partially ordered levels of the previous models using the concept of mathematical lattices.

***The Bell-La Padula confidentiality model***

This model classifies subjects and data into *sensitivity levels*. Orthogonal to these levels, *compartments* or *categories* are defined, which correspond to divisions or groupings within each level. The *classification*, C, of a data object defines its sensitivity. Similarly, users or subjects in general are given *clearance* levels. In each level an access matrix may further refine access control.

A *security level* is defined as a pair (classification level, set of categories). A security level *dominates* another (denoted as =>) if and only if its level is greater or equal than the other level and its categories include the other categories. Two properties, known as "no read up" and "no write down" properties, define secure flow of information:

**Simple security (ss) property**. A subject s may read object o only if its classification dominates the object's classification, i.e., C(s) => C(o). This is the no read-up property.

***-Property**. A subject s that can read object o is allowed to write object p only if the classification of p dominates the classification of o, i.e., C(p) => C(o). This is the no write-down property.

This model also includes *trusted subjects* that are allowed to violate the security model. These are necessary to perform administrative functions (e.g., declassify documents, increase a user's clearance). A pattern for the Bell-LaPadula model was given in [Fer01].

***The Biba integrity model***

Biba's model classifies the data into integrity levels (I) and defines two properties dual to the simple security and * properties:

**Single integrity property.** Subject s can modify object o only if I(s) => I(o).

**Integrity *-property.** If subject s has read access to object o with integrity level I(o), s can write object p only if I(o) =>I(p).

The first property establishes that an untrusted subject cannot write to objects of a higher level of integrity or she would degrade that object.

### 4.3 Problem

Many security attacks propagate through weak parts of a system. After finding an entry point, the malicious software may access a directory or some other unit of the architecture, frequently escalating its power because it will inherit rights from the compromised units. Protection rings can help with this problem but they can only be applied at the operating system level because of their need for hardware support.

The following **forces** affect the solution:

- Even if one part of the system is compromised or corrupted, other units will remain unaffected; that is, attacks or errors in a partition at some level should not propagate to other levels.
- The solution should be applicable to all architectural levels of the system, not just the operating system level.
- The solution should be applicable to distributed environments, not just single-processor systems.

### 4.4 Solution

Divide the architecture in such a way that transfers of control or data access from one division to another follows the Bell LaPadula and/or Biba restrictions. Assign functionality to levels according to their sensitivity.

***Structure***

Figure 7 shows a class diagram for the solution. A **Client Process** is assigned by the **System** to a specific **Partition** according to their function and degree of trust. A process can call other partitions for services according to a set of **Transition Rules**, based on a multilevel model.

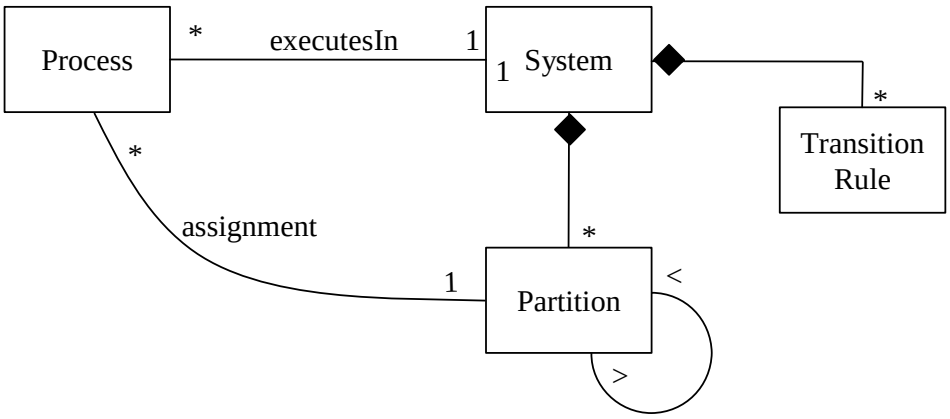


Figure 7. Class diagram for the Multilevel Secure Partitions pattern.

***Dynamics***
Figure 8 shows a sequence diagram for a process requesting a service from a different partition (p2), than the one where it is executing (p1). Requests for services in other partitions follow the rules of the multilevel model.

**4.5 Implementation**
The system must support a multilevel access control policy with mandatory restrictions, including data labeling and rule enforcement. Transitions from one partition to another should happen only through Protected Entry Points.

**4.6 Example resolved**
Figure 9 shows the solution provided by HP to the ChronOS problem [Rub94]. Now, if the web server is compromised, the attacker cannot deface web pages, cannot attack the web application server, cannot erase the log (they are all in different partitions).

**4.7 Known uses**
- HP's *Virtual Vault* [hpvv1, hpvv2, Rub94]—This is a secure platform for Internet applications that includes a trusted HP-UX operating system (a Unix variant), an enterprise server, firewalls, and other protection devices. It was part of a secure server system, the HP Praesidium family of products and appears to be the first use of this pattern. It also reduces the root privileges and controls inheritance of rights in forked threads (See Controlled Inheritance of Rights in [Fer02]).
- HP's restructuring of Unix' Sendmail program and other system programs [Zho98].
- HP's UX-11i, the current version of HP's operating system, also uses this approach [hpux].

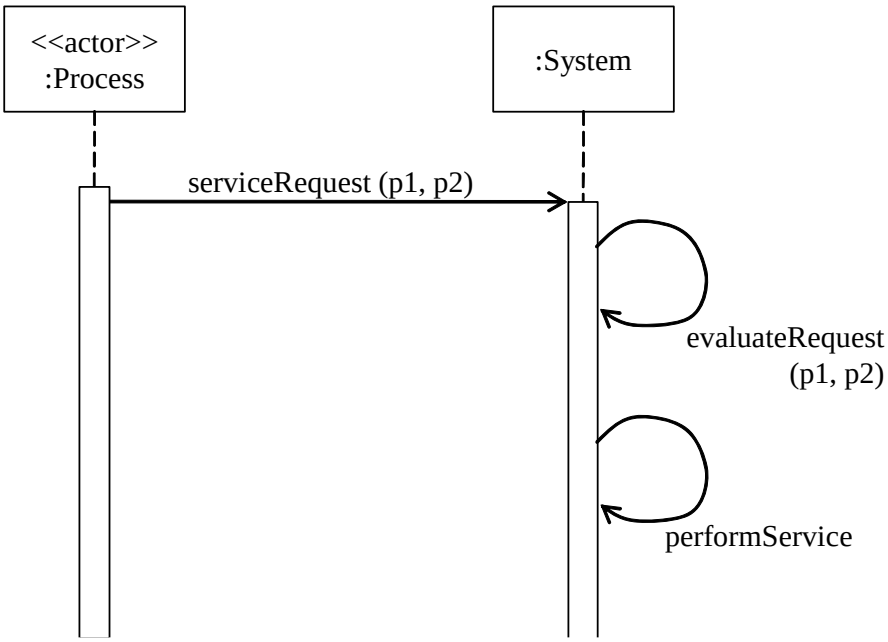


Figure 8. Sequence diagram for requesting a service within a partition.

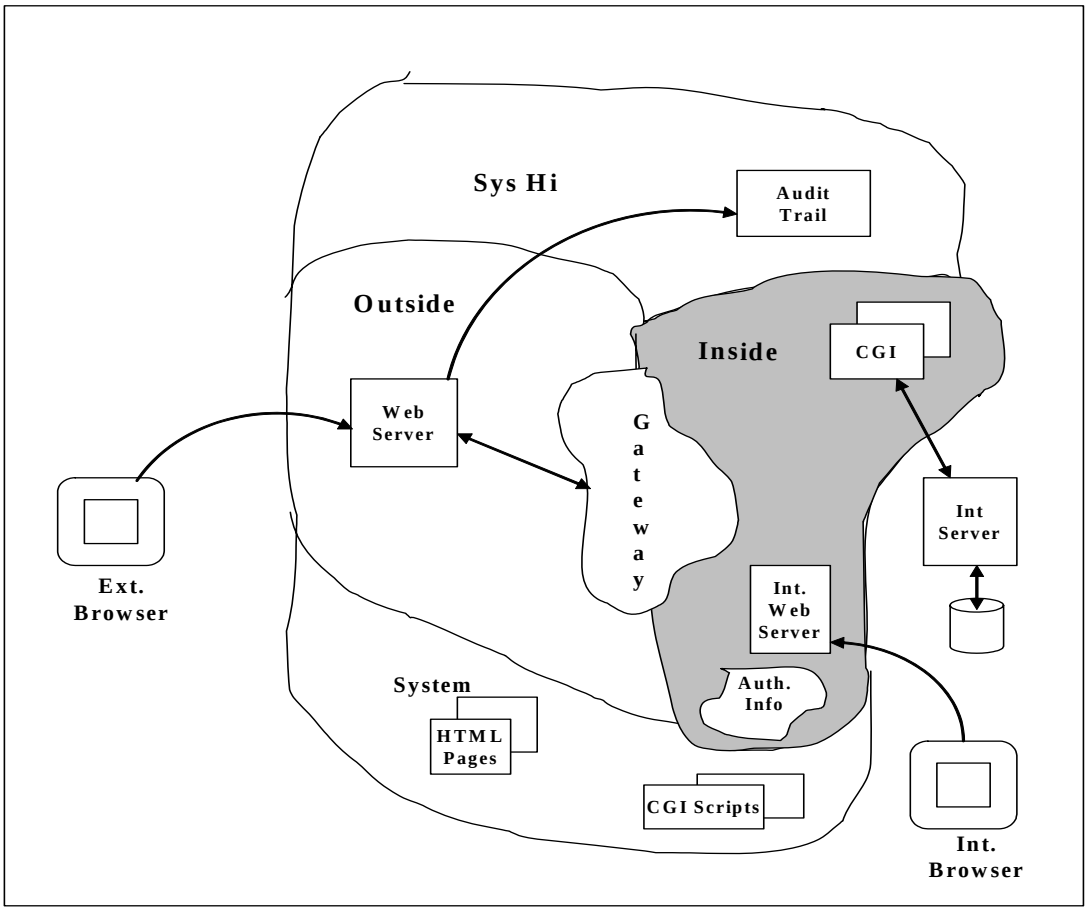


Figure 9. HP's Virtual Vault architecture

**4.8 Consequences**

This pattern has the following advantages:

- A compromised (taken over) or corrupted partition cannot propagate its attack or errors to other partitions.
- The solution does not depend on hardware support and can be applied in any architectural level, from applications to the operating system, to distribution units as in a Service-Oriented Architecture (SOA), .
- The solution is particularly suitable for distributed systems because it depends only on local attributes of the procedures and data involved.
- A multilevel model defines the relationship between subject and objects using a lattice model where it can be formally proven that the permitted accesses do not violate security restrictions. This only proves security in an abstract sense and could be affected by implementation details but it is a good security guideline for the complete system.
- Can lead to a more systematic definition of privileges because the levels can help in structuring them. This also makes administration simpler.

Possible disadvantages include:

- It may not be easy or even possible to place the required functions and data in the appropriate levels. This is particularly true at the application level, most commercial environments are not hierarchical.
- It is not simple to change the levels of existing programs or data. A trusted program is needed to override the rules and move subjects or data to other partitions [Gol06].

### 4.9 Related Patterns

- Multilevel Access Control pattern [Sch06]
- Protected Entry Points, see Section 2.
- Protection Rings, see Section 3.
- Protected Address Space (Sandbox) [Fer02]. The MSP pattern can only control the actions from a process with respect to other partitions, the Protected Address Space pattern can define precisely which resources within a partition can be accessed by the process.

## 5. Conclusions

We have presented three patterns that can improve the security and reliability of executing processes. Both the Protection Rings and the MSP patterns use Protected Entry Points to control changes of domain. While MSP applies to the control of executing processes in any level of the system and in distributed systems, Protection Rings require operating system and hardware support and only apply to centralized systems.

## Acknowledgements

We thank our shepherd, Neil Toussaint for his valuable suggestions that significantly improved this paper.

## References

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, Volume 1. J. Wiley, 1996.

[Day91] R.A.Dayan et al. *Signaling attempted transfer to protected entry point bios routine*, United States IBM (US) Patent 5063496, 1991, http://www.freepatentsonline.com/5063496.html

[Del07] N. Delessy. E.B.Fernandez, M.M. Larrondo-Petrie, and J. Wu, "Patterns for access control in distributed systems", *Procs. of the 14th Pattern Languages of Programs Conference (PLoP2007)*, Monticello, Illinois, USA, September 5-8, 2007 http://hillside.net/plop/2007/index.php?nav=program

[Fer78] E. B. Fernandez, R. Summers, T. Lang and C. Coleman, ``Architectural Support for System Protection and Database Security," *IEEE Trans. on Computers* , Vol. C-27, No. 8, pp. 767-771, August 1978.

[Fer02] E.B.Fernandez, "Patterns for operating systems access control", *Procs. of PLoP 2002*, http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

[Fer03] E. B. Fernandez and J. C. Sinibaldi, “More patterns for operating system access control”, *Proc. of the 8th European conference on Pattern Languages of Programs, EuroPLoP 2003*, http://hillside.net/europlop, 381-398.

[Fer05] E.B.Fernandez and T. Sorgente, "A pattern language for secure operating system architectures” , *Procs. of the 5th Latin American Conference on Pattern Languages of Programs*, Campos do Jordao, Brazil, August 16-19, 2005.

[Fer06] E.B.Fernandez, “Operating system access control”, Chapter 10 in [Sch06].

[Fro85] G. Frosini and B. Lazzerini, “Ring-protection mechanisms: general properties and significant implementations”, *IEE Procs*., vol. 132, Pt. E, No 4, July 1985, 203-210.

[Gol06] D. Gollmann, *Computer security (2nd Ed.)*, Wiley, 2006.

[Gra68] R.M.Graham, “Protection in an information processing utility”, *Comm. of the ACM*, vol. 11, No 5, May 1968, 365-369.

[hpux] HP-UX Operating system, http://en.wikipedia.org/wiki/HP-UX

[hpvv1] “VirtualVault. Philosophy of Protection”, 1998 http://docs.hp.com/en/B5413-90027/B5413-90027.pdf

[hpvv2] “What makes virtual vault secure?” http://www.docs.hp.com/en/J4255-90011/apas01.html#d0e11429

[int99] Intel Corp., *Intel Architecture Software Developer’s Manual, Vol. 3: System Programming.*

[mv] MV8000 principles of operation, http://cid-e17ca7e5bcaa1096.skydrive.live.com/self.aspx/P%c3%bablico/014-00648_MV8000_PrincOps_Apr80.pdf

[Oza88] B. M. Ozaki, E. B. Fernandez, and E. Gudes, ``Software Fault Tolerance in Architectures with Hierarchical Protection Levels, " *IEEE MICRO* , Vol. 8, No. 4, August 1988, 30-43.

[Rub94] C. Rubin and D. Arnovitz, “Virtual Vault white paper”, HP Praesidium, Hewlett Packard, 1994-1999.

[Sch06] M. Schumacher, E.B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*, J. Wiley & Sons, 2006.

[Shi00] T. Shinagawa, K. Kono, T. Masuda, “ *Exploiting Segmentation Mechanism for Protecting Against Malicious Mobile Code*”, Tech. Report 00-02, Dept. of Information Science, University of Tokyo, May 2000.

[Sil05] A. Silberschatz, P. Galvin, G. Gagne, *Operating System Concepts (7th Ed.)*, John Wiley & Sons, 2005

[Wal81] S. Wallach and C. Holland, “32-bit minicomputer achieves full 16-bit compatibility”, *Computer Design*, Jan. 1981, 111-120.

[wik] Wikipedia, “Ring (computer security)”, http://en.wikipedia.org/wiki/Supervisor_mode#Supervisor_mode

[Zho98] Q. Zhong and N. Edwards, “Security controls for COTS components”, *Computer*, June 1998, IEEE, 67-73.