A security pattern for Pipes and Filters

Eduardo B. Fernandez
Florida Atlantic University, Boca Raton, Florida, USA, ed@cse.fau.edu

Carlos Pertuz
Florida Atlantic University, Boca Raton, Florida, USA, cpertuz@fau.edu

Maria M. Larrondo-Petrie
Florida Atlantic University, Boca Raton, Florida, USA, maria@cse.fau.edu

ABSTRACT

Many applications need to process or transform a stream of data. Different stages are needed before data reaches the final stage. This happens for several reasons: every component performs specialized functions over the data, the global architecture or hierarchical organization requires this flow and this approach makes the system more flexible. Every time the data reaches a different stage, different functions may be applied at that stage. For example, in a law firm workflow, a secretary can create a legal document, but inserting legal advice or signing the document can only be done by lawyers. The Secure Pipes and Filters pattern provides secure handling of data streams. Each processing step applies some data transformation and the transformation and the movement of data can be controlled.

Keywords: Security, Patterns, Pipes, Filters

1. INTRODUCTION

Many applications need to process or transform a stream of data. Different stages are needed before data reaches the final stage. This happens for several reasons: every component performs specialized functions over the data, the global architecture or hierarchical organization requires this flow, and this approach makes the system more flexible. Every time the data reaches a different stage, different functions may be applied.

The architecture of this process is well defined and has been converted into a pattern.

Analysis, design, and architectural patterns are well established and have proved their value in helping to produce good quality software. Many companies have adopted their use as a way to improve their software. Security patterns are relatively new and starting to be accepted by industry because they are useful to guide the security design of systems by providing generic solutions that can prevent a variety of attacks [Sch06]. A pattern to describe the functional structure considered above is the Pipes and Filters [Bus96, Ort05]. However, that pattern considers only functional aspects; nonfunctional aspects such as security are not included. Security is needed when the stream includes sensitive or valuable data, such as legal documents or credit card files.

We present here the Secure Pipes and Filters, a secure version of the original pattern, which includes ways to add security controls at each stage of processing. This pattern is part of a catalog we are building to provide a variety of security mechanisms for different architectural levels. This catalog has its own value (its patterns can be used in isolation) but it is also part of a security systems development methodology we have proposed [Fer06a]. Section 2 presents the pattern following the template used in [Bus96]. Section 3 provides some conclusions and future work.

2. SECURE PIPES AND FILTERS

The Secure Pipes and Filters pattern provides secure handling of data streams. Each processing step applies some data transformation or filtering. The rights to perform the filtering and the movement of data can be controlled.

2.1 EXAMPLE

Suppose we are developing an application for a law firm [Fer07]. The type of work of such companies requires many documents to be created, filled, and approved. Before a document reaches its final version it has to be reviewed by different people. For instance a secretary creates a document following a general template, after which the document is given to an assistant lawyer who inserts technical aspects of the case, and finally the document goes to the principal lawyer, who decides the strategy to be followed in the legal action and signs the document as the legal representative. Because the operations on the document are not restricted we have serious problems. For example, a secretary made some changes to a legal document to help a friend that resulted in the firm collecting smaller fees than required.

2.2 CONTEXT

Systems where a stream of data needs to be handled in some sequential order.

2.3 PROBLEM

Some applications process or transform a stream of data [Bus96]. Different stages may be needed before data reaches the final stage. This happens for several reasons: every component performs specialized functions over the data, the global architecture or hierarchical organization requires this flow and this approach makes the system more flexible. Every time the data reaches a different stage, different functions may be applied. In the previous example, the secretary can create the legal document, but cannot insert legal advice or sign the document. In this kind of system, we may also need the flexibility to reorder the steps of the process or change the processing steps. In the example above, a new lawyer may be assigned to the case to perform additional functions on the documents which may require adding an extra step. How do we control the actions to be performed in a data pipeline?

The solution to this problem has to consider the following forces:

- The information can go in either direction in the system. Filtering may need to be applied when the data is moving in either direction.
- The system needs to assign privileges according to each stage of processing and the persons and tasks involved.
- We might require that before a document is accepted by the next stage, the previous stage or the message itself need to be authenticated.
- Due to regulatory constraints, work changes, or efficiency, some documents may need extra stages or skip stages. We need to be able to reconfigure the number or order of the steps. This reconfiguration must be controlled.
- For billing and security purposes, logging the actions at each stage may be necessary.
- The security controls should not affect the functional use of the system.

2.4 SOLUTION

The Pipes and Filters pattern provides a secure way to divide the processing of data into different sequential stages or steps. We add in each stage basic security mechanisms (as instances of security patterns) to provide authentication, authorization, and logging. Because the functions to be performed in each stage depend on persons doing specific tasks we use a Role-Based Access Control (RBAC) model [Fer01].

STRUCTURE

Figure 1 is an object diagram that shows the addition of security mechanisms to the Pipes and Filters pattern. The subsystems named **Authenticator** are instances of the Authenticator pattern [Sch06]. **Log i** indicates instances of the Logging pattern. Objects with the stereotype <<role>> and Right are instances of the RBAC pattern [Fer01, Sch06]. The **Reference Monitor** subsystem indicates the enforcement of authorization [Sch06]. We show the Reference Monitor as a shared resource and the Authenticators as individual for each stage; the actual distribution depends on the distribution architecture of the complete system. In order to control the reconfiguration of the stages, the RBAC pattern is also applied to the pipeline structure so that only someone with an administrator role (**Role3**) could perform any changes to it.

The RBAC pattern provides us with the option to abstract different roles within the data flow. It could be possible that we need to work with individual subjects instead of roles; in this case implementing the Access Matrix should be a better approach [Fer01]. The link between stages could be subject to attacks, and optional operations of encryption and decryption could be implemented in each filter, as well as digital signatures in each data message (not shown in the diagram).

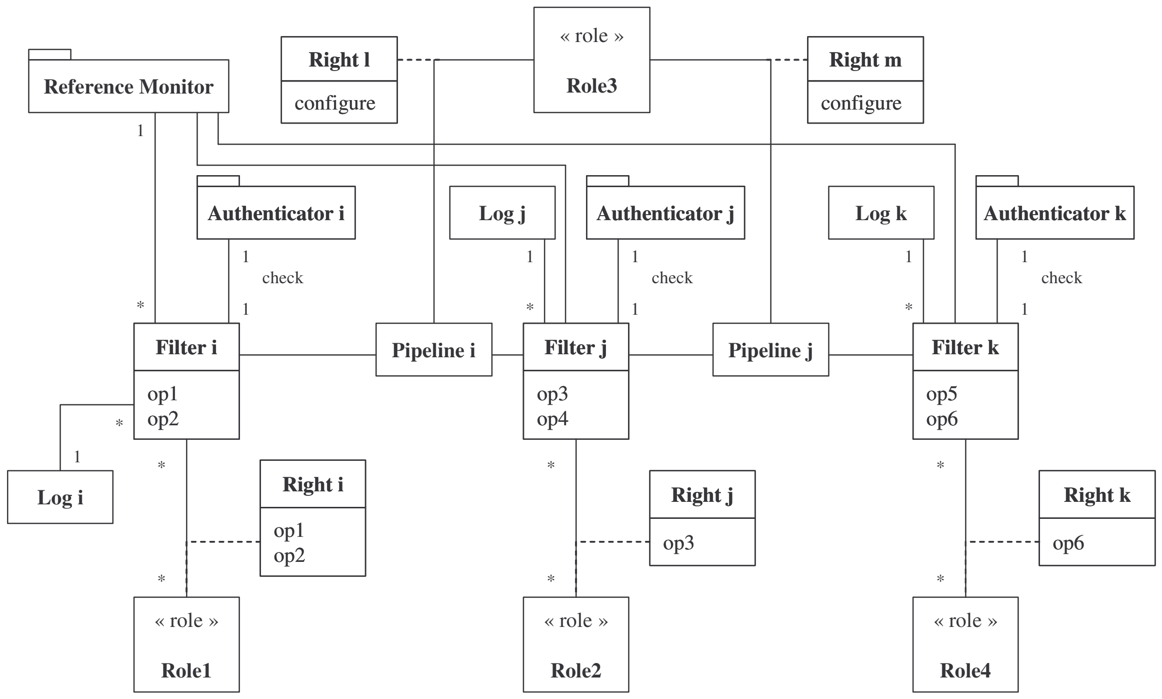

Figure 1. Object diagram for the Secure Pipes and Filters pattern

DYNAMICS

Figure 2 shows the use case where a subject with a specific role tries to execute an operation, op3, on a document. The Reference Monitor checks if its role allows the operation and if true it reads data from the input pipe, Pipe i, to the filter where op3 is applied. After the operation the data is moved to the next pipe, Pipe j.

2.5 IMPLEMENTATION

We followed the steps suggested in [Bus96] and indicate where security is needed:

1. *Divide the application into a sequence of stages.* As we discussed in [Fer06a], who should have access to which operations or results from each stage should be defined in the conceptual model. When the application is divided into stages we need to define how the rights in the complete model are reflected in each stage.
2. *Define the data format to be passed along each pipe.* This aspect has no effect on security.
3. *Decide how to implement each pipe connection.* Aspects such as active or passive components, push or pull movement of data are defined at this moment. At this moment we have to decide if we use or not authentication and if we do, what type of authentication.
4. *Design and implement the filters.* Each filter enforces the rights defined in the first implementation step and must have a Reference Monitor and a way to access the authorization rules. In distributed systems, one needs to decide where should these rules stored. Filters also implement logging.
5. *Design error handling.* From the security side this implies handling security violations. This handling is application-dependent and no general policy is possible.
6. *Set up the processing pipeline.* The initial or dynamic configurations must be restricted only to administrators.

2.6 EXAMPLE RESOLVED

We implemented RBAC in the Pipes and Filters of the example (Figure 3). Now the secretary, assistant and principal lawyers have specific rights over the document, according to their functions in the company. The right to reconfigure the stages within the data flow is controlled by the Administrator. Now a secretary could not make any changes to a document, she can only create the template for the document.

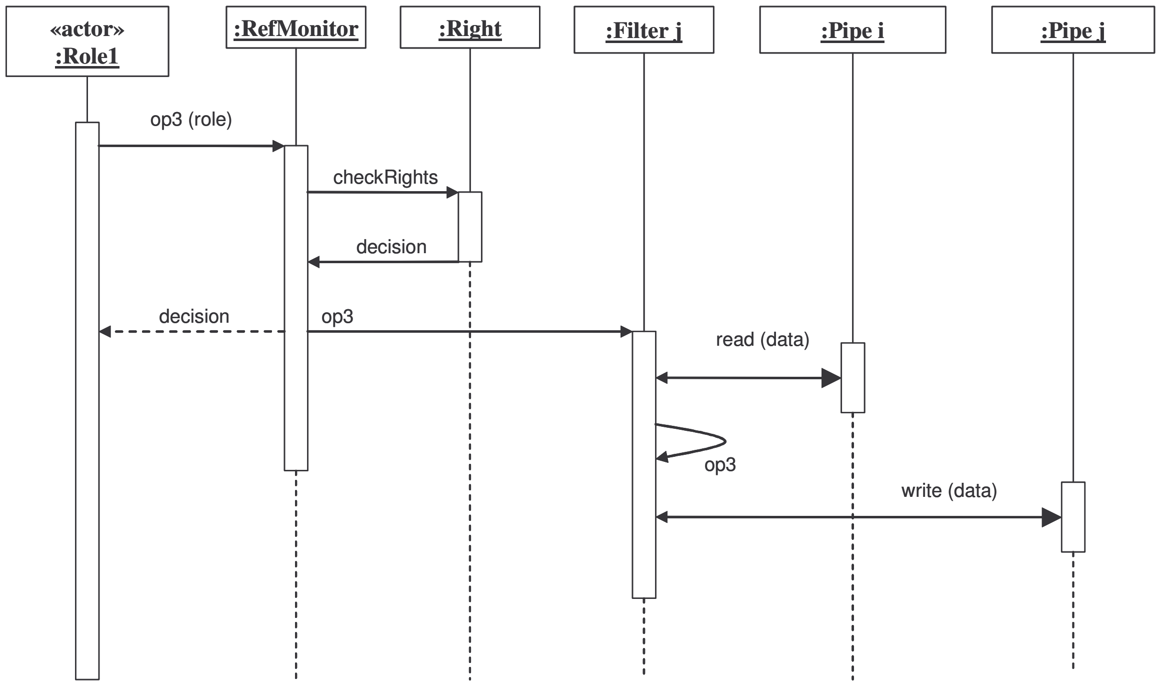

Figure 2. Sequence diagram for use case to apply an operation on a data stream

2.7 KNOWN USES

ROLE-BASED TRUST-MANAGEMENT MARKUP LANGUAGE (RTML) [LiN04]. This is an XML-based language to define RBAC models. It can be used to add rights to XML document pipelines.

MICROSOFT'S BIZTALK SERVER 2004 [Biz]. BizTalk Server 2004 can implement Pipes and Filters. In addition to security features that are provided by the transport, such as encryption when using HTTPS, BizTalk Server 2004 provides security at the message level. BizTalk Server 2004 can receive decrypted messages and validate digital signatures that are attached to these messages. Similarly, it can encrypt messages and attach digital signatures to messages before sending them.

INDUS [Ind]. This is an object oriented programming language for ubiquitous computing. It has syntax and semantics similar to Java but provides additional features such as: *Composition styles* – Indus defines syntax and semantics for several composition styles, e.g., pipes and filters, events, rules, etc, to enable plugging in of components by agents. *Security*: library support for security comprises different encryption schemes for location data, messages and mobile agent code.

COCOON [Coc07]. The Apache Cocoon is a web development framework using components. It can be used to build XML pipelines, including security restrictions.

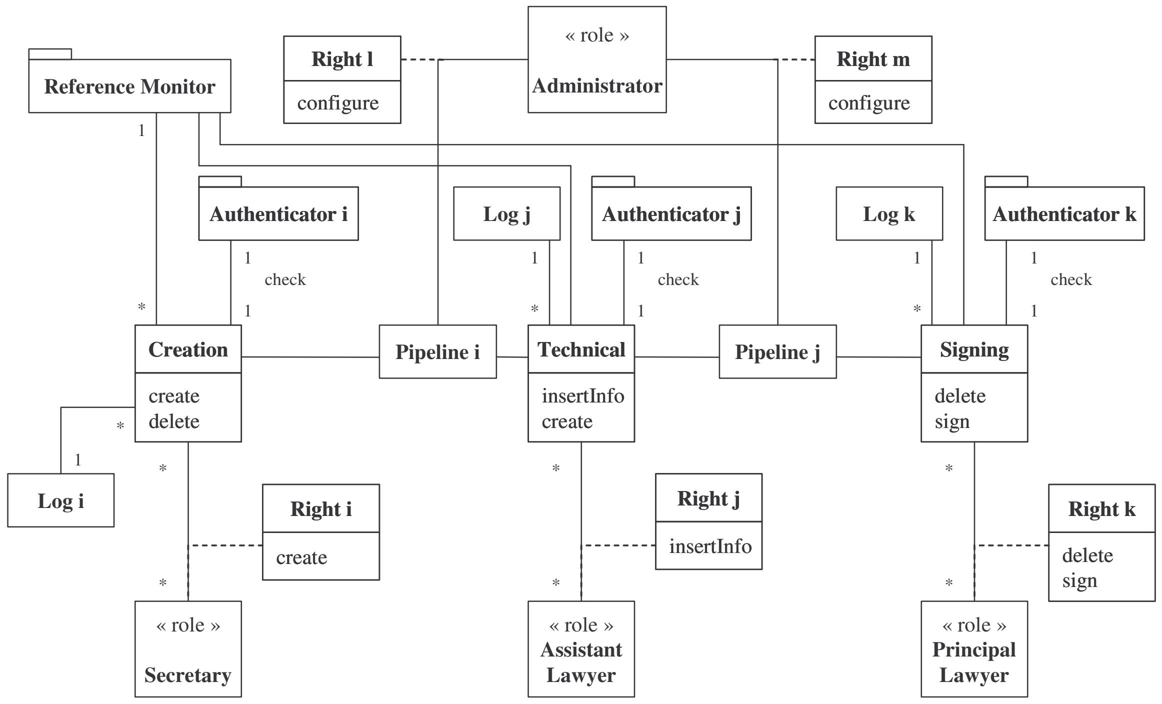

Figure 3. Secure Pipes and Filters applied

2.8 CONSEQUENCES

The use of this pattern yields the following benefits:

- We can assign privileges according to the functions needed at each stage of processing and the roles of those performing the functions. The use of operations over the data, is now restricted with the implementation of either RBAC or Access Matrix models.
- The use of encryption between stages is possible, adding the possibilities of secure messages and digital signatures.
- The Administrator role can control the reconfiguration of stages to accommodate changes in the process.
- Logging can be performed in each stage.
- The security restrictions are transparent to the users (while they do not attempt illegal actions).

Applying this pattern imposes the following liabilities:

- The general performance of the system worsens due to the overhead of the security checks.
- The system is more complex.

2.9 SEE ALSO

- The Layers pattern [Bus96] is considered a possible alternative for some uses of this pattern. The Secure Access Layers [Yod97] is a secure version of that pattern.
- The Access Matrix, RBAC, and Authenticator patterns can be used to secure the stages [Sch06].
- [Bus07] suggests implementing filters and pipes as instances of the Domain Object pattern.

3. CONCLUSIONS

The use of data pipelines or streams is frequent in software design. Following the principle that security must be applied in all stages of the software development, the designer should include security aspects in the pipelines in any application. As mentioned earlier, this pattern will become part of a catalog of security patterns. Combined with other similar patterns, it gives a designer a choice of possibilities when building the middleware of a complex system [Fer06b]. Future work includes developing secure versions of the Adapter and Blackboard patterns [Bus96], two patterns that are frequently used together with this pattern. Implementing these patterns together in a real application would be another useful direction that would confirm the value of using patterns; specifically implementing the law office example using web services and XML pipelines [Bro06].

ACKNOWLEDGEMENTS

We thank our Secure Systems Research Group (http://www.cse.fau.edu/~security) who provided valuable suggestions to improve this paper. The LACCEI reviewer also provided useful comments.

REFERENCES

[Biz] Implementing Pipes and Filters with BizTalk Server 2004 http://msdn2.microsoft.com/en-us/library/ms978668.aspx#implpipesandfilters_securityconsiderations

[Bro06] W. Brogden, "Using XML pipelines", Part 1: Sept. 6, 2006, Part 2: Oct. 3, 2006, http://searchwebservices.techtarget.com/home/0,289692,sid26,00.html

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

[Bus07] F. Buschmann, K. Henney, and D.C. Schmidt, *Pattern-Oriented Software Architecture, Vol. 4: "A Pattern Language for Distributed Computing,* J. Wiley & Sons, UK, 2007.

[Coc07] The Apache Cocoon project. http://cocoon.apache.org

[Fer01a] E. B. Fernandez and R. Pan," A Pattern Language for security models", *Procs. of the 8th Annual Conference on Pattern Languages of Programs (PLoP 2001),* 11-15 September 2001, Allerton Park Monticello, Illinois, USA, 2001. http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions

[Fer06a] E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "*Integrating security and software engineering: Advances and future vision",* H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.

[Fer06b] E. B. Fernandez and M. M. Larrondo-Petrie, "Developing secure architectures for middleware systems", *Procs. of CLEI 2006. (XXXII Conferencia Latinoamericana de Informática).*

E. B. Fernandez, D. L. laRed M., J. Forneron, V. E. Uribe, and G. Rodriguez G. "A secure analysis pattern for handling legal cases", accepted for the *6th Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP'2007).*

[Ind] Indus, an object oriented programming language for Ubiquitous computing https://developer.aumeganetworks.com/

http://en.wikipedia.org/wiki/Indus_programming_language

[Ort05] J.L. Ortega-Arjona, "The Pipes and Filters Pattern. A Functional Parallelism Architectural Pattern for Parallel Programming." *Procs.of the 10th European Conference on Pattern Languages of Programming and Computing (EuroPLoP 2005)*

[LiN04] N. Li, J.C.Mitchell, W.H. Winsborough, K.E. Seamons, M. Halcrow, and J. Jacobson, "RTML: A Role-Based Trust-Management Markup Language (RTML)", CERIAS Tech. Report 2004-03

https://www.cerias.purdue.edu/tools_and_resources/bibtex_archive/archive/2004-03.pdf

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

[Yod97] J. Yoder, J. Barcalow: "Architectural Patterns for Enabling Application Security", *Proc. of the 4tth Conference of Pattern Languages of Programs* (PLoP), 1997.Also, Chapter 15 in *Pattern Languages of Program Design*, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Boston, Massachussetts, USA: Addison-Wesley, 2000.