Securing the Adapter Pattern

Eduardo B. Fernandez and Jorge L. Ortega-Arjona

Abstract— **This paper presents a proposal for adding security issues to the well known Adapter pattern. Here, some possible attacks on the Adapter pattern are discussed, aproviding security approaches for such attacks. The objective is to obtain a level of security when applying this pattern into an application which makes use of the Adapter pattern.**

Index Terms— **Security, Adapter pattern**

I. Introduction

DESIGN and architectural patterns are now well-established for software development. There are a number of design and architectural patterns that have been written without considering their security aspects, including among others common patterns such as the Model-View-Controller (MVC) [2] and the Adapter [6]. We have proposed an approach to secure this type of patterns [3], [4]. A secured pattern can be used as a building block to develop an architectural infrastructure for secure applications. For example, [7] shows the use of this approach for securing the Broker pattern, [8] shows the securing of the Blackboard pattern, and [5] shows the securing of the Pipes and Filters pattern. We apply here this method to secure the Adapter pattern.

Section II presents a reworking of the original Adapter pattern to emphasize its traits of interest here. Section III discusses some possible attacks on the Adapter pattern and applies security policies to control these attacks. We end with some conclusions.

II. The Adapter Pattern

The Adapter pattern converts the interface of an existing class into a more convenient interface.

Also Known As

Wrapper

Eduardo B. Fernandez is with the Department of Computer Science and Engeneering, Florida Atlantic University, Boca Raton, FL 33431, USA. ed@cse.fau.edu

Jorge L. Ortega-Arjona is with the Departamento de Matemáticas, Facultad de Ciencias, UNAM. México D.F. 04510, MEXICO. jloa@fciencias.unam.mx

Example

We have a text message system that sends, receives, and manipulates text messages. We want to convert our text messages into XML messages so that we can handle more complex transactions. We purchased an off-the-shelf tool, XmlMessage, which manipulates XML messages. The problem is that these two interfaces are incompatible: XmlMessage expects an XML message and TextMessage does not know how to create an XML message.

Context

A computational environment where users or processes need to use a class which has an interface that is incompatible with the current class.

Problem

We need to convert the interface of a class into another class.

Forces

- The existing class has useful functions; if we cannot find a way to use it we will need to rewrite all these functions.
- We need to decide how much the adapter should adapt. There may be many functions and we only need a few of them.

Solution

Define an Adapter class that adapts some functionality of the class been adapted. We can do this using class Adapters and object Adapters. We show the details of an object Adapter, the class Adapter can be seen in [6].

Structure

Figure 1 shows the structure of an object `Adapter`. The `Client` sends a `request()` to the `Target`. The `request()` defined in class `Target` is converted by the `Adapter` into a `specificRequest()` defined in the class `Adaptee`. The `Adapter` can find the `Adaptee` through an association.

Dynamics

Figure 2 shows a sequence diagram of a `request()` been adapted to a `specificRequest()`. A `Client` sends a `request()` to the `Target`, which delegates it to the `Adapter`. The `Adapter` is responsible to

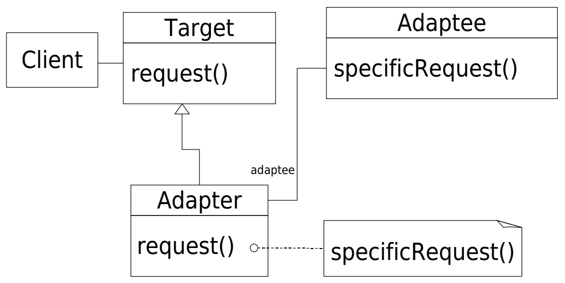

Fig. 1. Adapter Class Diagram.

convert such `request()` to a `specificRequest()` defined in the class `Adaptee`. The `response` is then returned to the client.

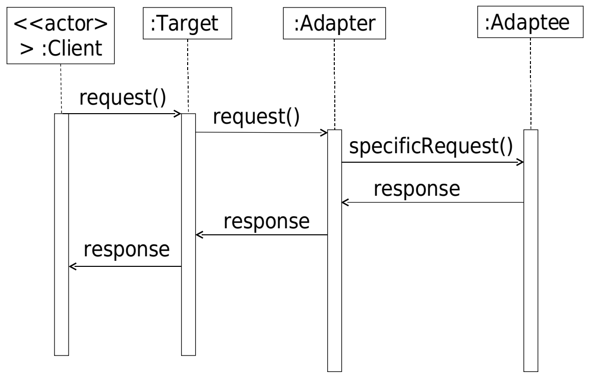

Fig. 2. Sequence Diagram for the Adapter.

Example resolved

We can create an adapter class, `Message`, which receives all requests to create XML messages and returns XML messages. The text message is structured in a certain format; for example, sender's id, location, name, message and so forth. We use this format to create the XML message. Figure 3 shows the class diagram for the solution.

Consequences

Class Adapter advantages include:

- A class Adapter adapts the interface of an existing class into a more suitable interface.
- Since the Adapter class is a subclass of the Adaptee class, it can override some of the adaptee's behavior.
- No additional pointer indirection is needed to get to the adaptee.

Class Adapter disadvantage include:

- A class adapter will not work when we want to adapt a class and all its subclasses.

Object Adapter advantages include:

- It lets a single Adapter work with many adaptees and it can add functionality to all adaptees at once.

Object Adapter disadvantages include:

- An object Adapter makes it more difficult to override adaptee behavior.

An advantage of both varieties is that if the Adaptee is a Facade of a complex system, e.g. a relational database, we can define a uniform interface for application programs to use the complex system.

Related Patterns

Bridge, which separates an interface from its implementation; Decorator, which enhances another object without changing its interface; Proxy, which acts as a surrogate for another object and does not change its interface [6]; and Wrapper Facade, which encapsulates low-level functions and data structures within higher level interfaces [9].

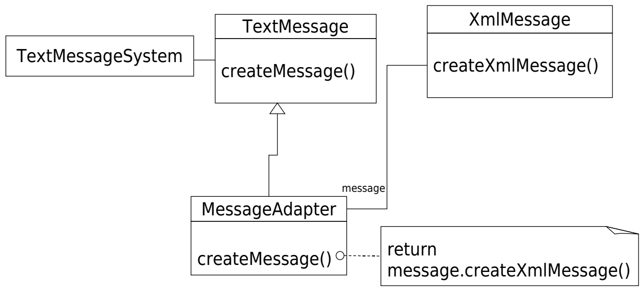

Fig. 3. Adapter Class Diagram for Text and Xml Messages Example.

III. Adding Security to the Adapter Pattern

We consider now some possible attacks to the adapter and provide some solutions to avoid such attacks.

A. Attacks on the Adapter

As shown earlier, the Adapter converts the interface of an existing class into a more convenient interface, but its original description does not take in consideration security issues.

To illustrate and identify some possible attacks, let us consider the following example: We have an interface, RequestServices, which is used to request services from some servers. We now want to be able to send requests to a JDBC API; however, our interface is incompatible with the JDBC API. We create a RequestServicesAdapter that adapts requests to JDBC. For example, a client sends a request for a database connection. The RequestServicesAdapter converts the request to a JDBC request, which in turn returns a response containing the requested data items.

We can identify the following threats in this example:

- **T1**. The database accessed through the JDBC interface could be an impostor and we could be sending or receiving data from a malicious database.
- **T2**. The client may be an impostor, trying to access the data of an authorized user.
- **T3**. The client making the request may not have permission to send such request; i.e. the client may try to access data to which it is not authorized.
- **T4**. If the client is remote, the data sent and received may be intercepted by intruders.

After we identify the possible threats to the Adapter we need to define policies and their corresponding mechanisms to stop them:

- **T1**. Authenticate the database.
- **T2**.Authenticate the client.
- **T3**. Control access to the Adaptee functions through the Adapter.
- **T4**. Add a secure channel between the client and the Adapter.

Figure 4 shows a class diagram for the Secure Adapter. We add Role-Based Access Control for the clients and a corresponding set of authorization rules. Requests made to the Adapter have to be authorized ensuring that the client has permission to send such requests. The Adapter also checks responses returned by the Adaptee. For example, when a client requests a database connection, the Adapter authenticates the database identity returned in the response from the Adaptee. The secure channel is not shown in the figure.

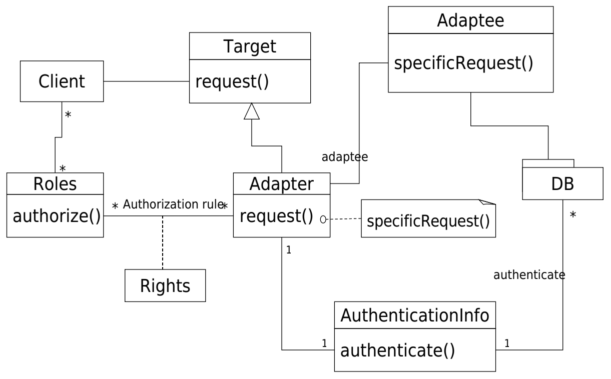

Fig. 4. Class Diagram for the Secure Adapter.

Figure 5 shows a sequence diagram for the use case Request data in the secure Adapter. The client sends a request to the Target. Such request is captured by the Adapter, which is responsible to authorize the client. Once the client's permission is verified, the Adapter converts the request to a specific request. The Adaptee renders such request and sends a response back to the Adapter. At this point the Adapter needs to make sure that the identity of the subject in the response is not an imposter. After authenticating the response subject, the Adapter sends the response to the client.

B. Known uses

CORBA-based systems use Adaptors to adapt a remote request to the servant object [9]. The Adaptor also applies authorization constraints.

Microkernels use adaptors to adapt process requests that might have different formats [2]. This makes the microkernel more reusable. In some implementations the adaptor applies authorization restrictions.

C. Related Patterns

Role-Based Access Control [10]. How do we assign rights to people based on their functions or tasks? Assign people to roles and give rights to these roles so they can perform their tasks.

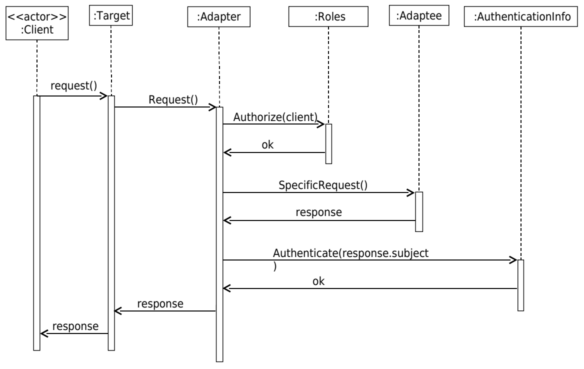

Fig. 5. Sequence Diagram of the Secure Adapter.

Authenticator [10]. How to verify that a subject is who it says it is? Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject.

Secure Logger [11]. How to capture the application-specific events and exceptions in a secure and reliable manner to support security auditing?

Secure Channel [1]. Define a secure communication channel between two remote processes.

Microkernel [2]. Microkernels use adapters to adapt to different types of client requests.

IV. Conclusions

We have examined the Adapter pattern and considered some possible attacks in order to add security to it. Our approach was first to describe the Adapter in detail and then evaluate security issues. We applied a method we proposed earlier [3], [4]. We conclude that the two main problems are: the client making a request may not have permission to do so; and that the subject in the response may be an impostor. In both cases we could compromise the integrity of our application, lose data, and facilitate the misuse of confidential information. To solve these problems we need to check the authorization rights of the client and authenticate the subject returned in the response.

To complement this work we can consider some specific implementation and check for additional security concerns introduced by the lower levels. Another future work is to secure the MVC pattern.

References

[1] A. Braga, C. Rubira, and R. Dahab, *Tropyc: A pattern language for cryptographic object-oriented software*, Chapter 16 in Pattern Languages of Program Design 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in Procs. of PLoP'98.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal *Pattern-Oriented Software Architecture. A System of Patterns*, New York: John Wiley & Sons, 1996.

[3] E.B. Fernandez, S. Huang, and M.M. Larrondo-Petrie *Building secure middleware using patterns*, Procs. of the IEEE Int. Symposium and School on Advanced Distributed Systems (ISSADS 2006), January 2006.

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

[5] E.B Fernandez and J.L. Ortega-Arjona, *The Secure Pipes and Filters pattern*, accepted for the Third Int Workshop on Secure System Methodologies using Patterns (SPattern 2009), Linz, Austria, Sept. 2009.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides *Design Patterns. Elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wesley, 1995.

[7] P. Morrison and E.B. Fernandez, *Securing the Broker pattern*, Procs. of the 11th European Conf. on Pattern Languages of Programs (EuroPLoP 2006) `http://www.hillside.net/europlop/`

[8] J.L. Ortega-Arjona and E.B. Fernandez, *The Secure Blackboard pattern*, Procs. of the 15th Int.Conference on Pattern Languages of Programs (PLoP 2008), co-located with OOPSLA, Nashville, TN, Oct. 2008.

[9] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-oriented software architecture, vol. 2, Patterns for concurrent and networked objects*, J. Wiley, 2000.

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

[11] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Strategies for J2EE Web Services, and Identity Management*, Prentice Hall, Upper Saddle River, New Jersey, 2005.