Security Considerations

Security considerations associated with the Implementing Message Replay Detection in WSE 3.0 pattern include the following:

- You must set the cache lifetime for the custom policy assertion for a longer time than the maximum message age configured in the policy assertion added to twice the WSE 3.0 configuration value for time tolerance in seconds. This should not depend on the expiration of the message specified by the sender.
- Replay caches do not inherently provide a means for a service to detect cache tampering. If replay cache tampering is an identified threat that you choose to mitigate, as revealed by a proper threat analysis of your application, consider requiring the service (or services on a Web farm) to create a Hashed Message Authentication Code (HMAC) or digital signature on the cache contents to verify the cache's integrity. This approach is effective to mitigate cache tampering, but it also degrades the performance of the replay detection mechanism.
- For simplicity, the examples in this pattern do not apply mitigation techniques against all possible threats. For example, all input should be validated. For more information, see Message Validator in Chapter 5, "Service Boundary Protection Patterns."
- If you are using a perimeter service router to route the same types of messages to several different service endpoints, you have to make sure that a service will not process a replayed message that was already processed by one of the other service endpoints that receives messages from the router. To mitigate a message replay across multiple services, you must either make sure that the replay cache is shared by all the services or implement message replay detection on the perimeter service router.

Message Validator

Context

A Web service interacts with other applications over a network. Incoming data may be malformed and may have been transmitted for malicious purposes. There is also a risk of injection attacks, where data from incoming messages is tampered with to include additional syntax.

Problem

How do you protect Web services from malformed or malicious content?

Forces

Any of the following conditions justifies using the solution described in this pattern:

- **Malicious content poses a risk to the Web service**. An attacker can insert syntax in a request message to cause the Web service or other downstream systems that process the received data to behave in an undesirable manner. The attacker can do this through injection attacks, such as XML injection, SQL injection, or HTML/client script injection. Web services that do not require access control are especially susceptible because they have no means to limit to a smaller, more trusted group, the number of clients that can access them.
- **There is a risk of attackers bypassing client validation techniques by using alternative clients or by modifying data after it has left the client**. Web services must be designed to be autonomous and perform their own input validation instead of trusting the validation that is performed in the client application.

The following condition is an additional reason to use the solution:

- **An attacker can use malformed or oversized messages to launch a denial-of-service attack**. Denial of service attacks can take advantage of the multiplier effect, where a malformed or oversized message causes a disproportionate increase in the use of resources, such as a server's CPU time, memory usage, or database connections.

Solution

Assume that all input data is malicious until proven otherwise, and use message validation to protect against input attacks, such as SQL injection, buffer overflows, and other types of attacks. The message validation logic enforces a well-defined policy that specifies which parts of a request message are required for the Web service to successfully process it. It validates the XML message payloads against an XML schema (XSD) to ensure that they are well-formed and consistent with what the Web service expects to process. The validation logic also measures the messages against certain criteria by examining the message size, the message content, and the character sets that are used. Any message that does not meet the criteria is rejected.

Participants

Message validation involves the following participants:

- **Client**. The client accesses the Web service.
- **Service**. The service is the Web service that processes requests received from clients. The service implements the message validation logic.

Process

Figure 5.4 illustrates the process that is used by message validation logic to intercept request messages and verify that they are acceptable for processing by the service.

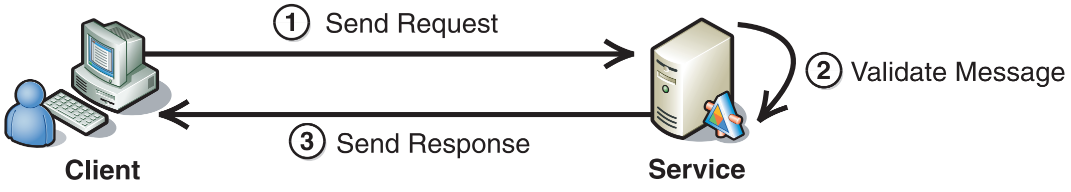

Figure 5.4
Message validation occurring at a Web service

As illustrated in Figure 5.4, the process for message validation is described in the following steps:

1. **The client sends a request message to the service**. The validation process itself is hidden from the client.
2. **The service validates the message**. The message validation logic makes a number of checks to validate the message. Checks can include:
   - Comparing the size of the request against the maximum allowable size that is specified for request messages.
   - If the message is signed, verifying the signature to ensure that the message has not been tampered with in transit.
   - Verifying that the message payload is well-formed and conforms to a predefined schema, with acceptable data types and ranges of values.
   - Parsing the entire request message for malicious content. Potentially, malicious content can be placed in either the SOAP message elements or in the message payload, so both are checked.
3. **The service processes the request and responds to the client**. If the request passes all the validation checks that are performed by the message validator, the service processes the message and may issue a response to the client.

Resulting Context

This section describes some of the more significant benefits, liabilities, and security considerations of using this pattern.

Note: The information in this section is not intended to be comprehensive. However, it does discuss many of the issues that are most commonly encountered for this pattern.

Benefits

The benefits of the Message Validator pattern include the following:

- The Web service is protected from malformed and malicious content. This helps protect against injection attacks, even for Web services that do not implement access control.
- The Web service performs validation independently of the client — it does not accept messages simply because they have already been validated by the client.

Liabilities

The liabilities associated with the Message Validator pattern include the following:

- Message validation logic does not process binary message content, such as attachments. For message validation logic to process binary attachments, it needs to be capable of recognizing each type of binary attachment that it encounters to ensure that they are free of malicious content. Specifying a maximum message size helps to protect against injection attacks in binary attachments. However, validation of binary data should be handled by antivirus filters.
- If a message is encrypted with message layer security, it may not be possible to inspect data for malicious content unless the message is decrypted beforehand or the validation logic has access to the decryption key.
- If data is protected by transport layer security, the entire channel is encrypted and decrypted at end points. As a result, message validation cannot occur at any intermediaries between those points.

Security Considerations

Security considerations associated with the Message Validator pattern include the following:

- Message validation can help protect against denial of service attacks, but the message validation logic must be very efficient when it conducts its validation checks. Otherwise, the message validation logic may be a system bottleneck and may itself become the target of a denial of service attack. Malformed content can include very large messages, in some cases for the purposes of launching a denial of service attack. You should make the maximum message size large enough to allow legitimate messages to be accepted but small enough to prevent attacks.
- Using a validating parser and verifying the input message against its XML Schema (XSD) result in a significant increase in CPU processing. And, even though XML Schema (XSD) has the capability to specify data range validations and it supports the use of regular expressions, many schemas use data types, such as string, which do not prevent many forms of injection attacks.

- Instead of building the message validation logic into the Web service itself, you can place it in an intermediary. This allows several Web services to use the same intermediary, and it enables each Web service to dedicate its resources to processing legitimate messages. It also ensures that invalid messages never reach the Web service. However, using an intermediary in this way can create a single point of failure, which may become a target of attack.
- XML message payloads that contain a CDATA field can be used to inject illegal characters that are ignored by the XML parser. If CDATA fields are necessary, you must inspect them for malicious content.
- The Web service may obtain data for response messages from external sources. There is no guarantee that external data sources properly validate data. Passing responses without message validation makes the Web service a potential "carrier" of malicious input from external data sources. You should consider filtering Web service response messages that are returned to the client.

Related Patterns

The following child pattern is related to the Message Validation pattern:

- **Implementing Message Validation in WSE 3.0**. This pattern provides steps and recommendations to implement message validation at the message layer with WSE 3.0.

Implementing Message Validation in WSE 3.0

Context

You are implementing a Web service that uses Web Service Enhancements (WSE) 3.0. The Web service must validate request messages received from clients to make sure that they are not malformed and do not contain malicious content.

Objectives

This implementation of the Message Validator pattern has the following objectives:

- Prevent the service from processing request messages that are larger than a specified size.
- Prevent the service from processing messages that are not well-formed or that do not conform to an expected XML schema.
- Validate input messages before deserializing them into .NET data types so that they can be interpreted as regular expressions.
- Demonstrate how to use WSE 3.0 custom assertion to implement message validation.
- Use ASP.NET and WSE 3.0 configuration settings to limit usage of system resources such as CPU.