Exception Shielding

Context

A client is accessing a Web service. The Web service is designed according to the principals of service orientation, which ensures that the boundaries of the service are explicit, and requires that exception information related to the internal implementation of the service is managed within the service.

Problem

How do you prevent a Web service from disclosing information about the internal implementation of the service when an exception occurs?

Forces

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

- **Exception details may contain clues that an attacker can use to exploit resources used by the system**. Detailed fault messages can disclose information about the Web service or resources accessed by the Web service code that threw the exception. An attacker may deliberately cause the Web service to throw an unhandled exception in an attempt to obtain sensitive information, such as connection strings, server names, SQL queries, XPath commands, stack traces, and data schemas. The attacker can then use this information to exploit the Web service or the resources that it accesses.
- **Information related to anticipated exceptions needs to be returned to the client.** In cases where an exception is expected, an error message that does not contain sensitive internal information can be returned to the client. A service may provide information about the cause of the fault, where the information is not considered a security risk. In some cases (for example, data validation errors), the potential savings in administrative support may outweigh the risk of providing the requestor with more detailed information about an exception.

The following condition is not resolved by the base pattern, but it is resolved by Extension 1 — Logging Exceptions:

- **Exceptions that occur within a Web service should be logged to support troubleshooting**. Information within an exception can be used by monitoring tools to automatically notify system administrators when an exception occurs. The same information can also be used by application developers to diagnose exceptions that occur within the logic of the service or with resources that the service is dependent on. In some cases, you may require that an error message that is returned to the client contains an ID that helpdesk staff can use to troubleshoot user problems.

For more information, see the “Extensions” section at the end of this pattern.

Solution

Use the Exception Shielding pattern to sanitize unsafe exceptions by replacing them with exceptions that are safe by design. Return only those exceptions to the client that have been sanitized or exceptions that are safe by design. Exceptions that are safe by design do not contain sensitive information in the exception message, and they do not contain a detailed stack trace, either of which might reveal sensitive information about the Web service's inner workings.

Participants

Exception shielding involves the following participants:

- **Client**. The client application that calls a Web service.
- **Service**. The Web service that processes requests that are received from clients.

Process

Figure 5.6 illustrates how an unhandled exception that is thrown by a Web service is processed by a service that implements exception shielding.

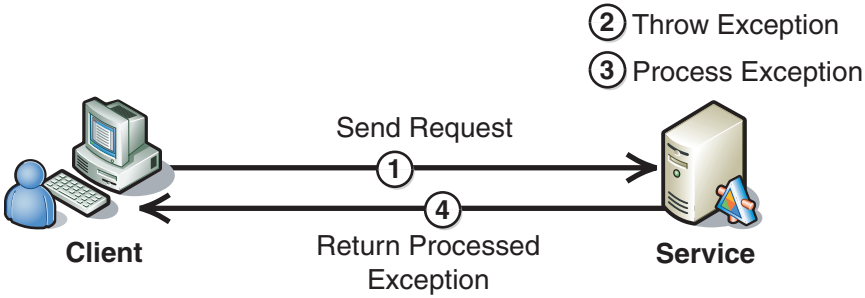

Figure 5.6
A Web service that implements exception shielding

As illustrated in Figure 5.6, the exception shielding process is described in the following steps:

1. **The client submits a request to the service**.
2. **The service attempts to process the request and throws an exception**. The exception can be safe or unsafe by design.
3. **Exception shielding logic processes the exception**. If the exception type is safe by design, it is already considered sanitized and is returned to the client unmodified. If the exception is unsafe, the exception is replaced with an exception that is safe by design, which is returned to the client.
4. **The service returns the processed exception to the client**. The exception is wrapped in a SOAP fault before it is returned to the client.

Example

Global Bank has designed a Web service that checks the balance of customer accounts. Global Bank needs to ensure that when exceptions occur, information potentially useful to attackers is not revealed.

For some anticipated exceptions that are safe by design, such as data validation errors, the Web service returns appropriate information to the client. For other exceptions, such as authentication failures, the exception logic sanitizes the exception, replacing it with an exception that is safe by design.

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 using the Exception Shielding pattern include the following:

- Exception shielding prevents sensitive information from being disclosed in exception details.
- Maintenance staff can enable detailed exception information to be returned by production Web services. This allows them to troubleshoot issues in the production environment without exposing exception details to external consumers.
- Unanticipated exceptions that are thrown by Web services in the enterprise can be uniformly and centrally managed. Different Web services that implement disparate methods of exception management make it more difficult for enterprise architects to ensure that unhandled exceptions are managed securely and consistently across an enterprise.

Liabilities

Adding exception shielding logic to a Web service increases the amount of processing the service must perform. You must ensure that exception shielding is performed efficiently. Any related activities, such as logging, may need to be minimized to prevent the service from becoming a performance bottleneck.

Security Considerations

Security considerations associated with the Exception Shielding pattern include the following:

- Unhandled exceptions may be wrapped by another exception. You should ensure that the outer exception and all wrapped exceptions are checked by the exception shield logic before they are returned to a Web service client.
- You should use exception handling throughout the entire application's code base. This prevents internal implementation details of the service from being revealed to the client.
- The "deny" model is an alternative to the "allow" model that is used in the Exception Shielding pattern. In the deny model, specific exceptions are registered to be sanitized, and all other exceptions are sent back to the client unmodified. However, the deny model is considered less secure, because unanticipated exceptions are not sanitized.

Extensions

The extension described here builds on the base pattern to provide additional capabilities. In addition to resolving the forces stated for the base pattern, this extension also resolves the following condition:

- **Exceptions that occur within a Web service should be logged to support troubleshooting**. Information in an exception can be used by monitoring tools to automatically notify system administrators when an exception occurs. The same information can also be used by application developers to diagnose exceptions that occur within the logic of the service or with resources that the service is dependent on. In some cases, you may require that an error message that is returned to the client contains an ID that helpdesk staff can use to troubleshoot user problems.

Extension 1 — Logging Exceptions

In addition to processing exceptions, the exception shielding logic can also log the full details of the exception to an event log. This allows maintenance staff to identify and troubleshoot the exceptions. The information also assists with intrusion detection and incident response.

The exception shielding logic can also generate an exception identifier for each exception and pass it back to the client in a message, so that it can be presented to the user in the form of an error message. This allows the exception that is returned to the client to be directly traced to detailed exception information located in the event log, which can assist in dealing with helpdesk calls.

Related Patterns

The following child pattern is related to the Exception Shielding pattern:

- **Implementing Exception Shielding**. This pattern provides implementation steps and recommendations for using exception shielding.

Implementing Exception Shielding

Context

You are implementing a Web service that runs on the .NET Framework. You must ensure that exceptions thrown by the Web service do not disclose sensitive information about the service or resources that it accesses.

Objectives

The objectives of this pattern are to:

- Prevent the Web service from disclosing sensitive information in exception messages.
- Create exceptions that are safe by design in which exception information is returned to Web service clients.
- Write unsanitized exception details to a log to support monitoring and troubleshooting the Web service.

Content

This pattern consists of the following sections:

- **Implementation Strategy**. This section provides a high-level description of the strategy used to implement the solution that includes descriptions of the participants and the process.
- **Implementation Approach**. This section describes the following steps that are required to implement the Exception Shielding pattern:
  - Create a custom exception class.
  - Enclose code in **try/catch** blocks.
  - Create a method that sanitizes exceptions.
- **Resulting Context**. This section outlines the benefits, liabilities, and security considerations when the pattern is implemented.

Note: The code examples in this pattern are also available as executable QuickStarts on the Web Service Security community workspace.
