Message Replay Detection

Context

A client calls a Web service by sending messages across a public network. When the Web service processes the messages, data is updated or business processes are initiated. If a message that is intended for one of these Web services is intercepted and replayed, there is a risk that the same operation might be performed multiple times.

Problem

How do you protect a service from an attacker who replays an intercepted message?

Forces

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

- **A replayed message will cause data inconsistency**. This can have a negative impact on business operations and cause financial damage or legal liability. For example, if funds are transferred between bank accounts multiple times, the balance of each party's account will be altered.
- **Messages traverse intermediaries on the network, where the intermediaries are not trusted**. When messages traverse untrusted intermediaries, they can be intercepted and replayed after the initial relay of the message. This form of attack is possible even if message protection techniques, such as data origin authentication and data encryption, are used to protect against tampering of data and unauthorized access to data.

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

- **The Web service is susceptible to message flooding denial of service attacks from message replay**. If the normal functions of a Web service are system-intensive or network-intensive, an attacker can cause a bottleneck in the service by launching an automated attack that rapidly replays intercepted messages in large quantities. This reduces the availability of the service.

Solution

Cache an identifier for incoming messages, and use message replay detection to identify and reject messages that match an entry in the replay detection cache.

Message replay detection requires that individual messages can be uniquely identified. This ensures that a legitimate message is not rejected because of a match in the replay detection cache. Message replay detection also requires that messages have not been tampered with in transit. This ensures that the replay detection cache does not accept messages that have been captured and modified by an attacker.

Messages signed using a WS-Security XML signature must include a **SignatureValue** element, which can be cached as an identifier for the message. The **SignatureValue** is computed from hash values of the message parts that are being signed, including the message body and the timestamp.

Note: A **SignatureValue** is not truly unique because it runs the theoretical risk of collision (where the same value can be unintentionally reproduced). In most cases, the risk is very low, so **SignatureValue** is an appropriate choice for a message identifier.

The **SignatureValue** element is added to the cache, along with a timestamp from the server, indicating the time it processed the message. This allows entries to be cleared from the cache at regular intervals and to not accumulate indefinitely. The service can be designed to automatically reject incoming messages that arrive on or after a defined acceptable time delay.

Participants

The Message Replay Detection pattern involves the following participants:

- **Client**. The client accesses the Web service.
- **Service**. The service is the Web service processes requests received from clients. The service implements the replay detection logic.
- **Replay cache**. The replay cache is the entity that caches the incoming messages with a unique identifier to detect the replay messages.

Process

Figure 5.1 illustrates the process of sending a message to a Web service that implements replay detection.

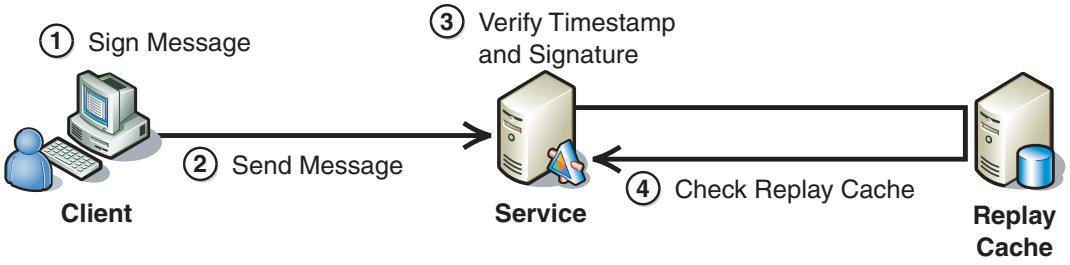

Figure 5.1
A Web service implementing message replay detection

As illustrated in Figure 5.1, the process of a Web service implementing message replay detection is described in the following steps:

1. **The client signs the message**. This signature provides assurance that the message has not been altered in transit. For more information about data integrity, see Data Origin Authentication in Chapter 2, "Message Protection Patterns."
2. **The client sends the signed message to the recipient**.
3. **The service verifies the client's signature and the message timestamp**. The Web service verifies the message signature to ensure that the message contents have not been altered in transit. If the message signature is valid, the Web service compares the message timestamp to its own current clock value. If either the signature is invalid or the message was received beyond the acceptable time span, the message is rejected.
4. **The service checks the replay cache for the SignatureValue field**. The Web service checks the replay cache for the **SignatureValue** that is used to uniquely identify the incoming message. If the **SignatureValue** is already in the cache, the message is rejected as a duplicate. If the message signature is not in the cache, the message signature and timestamp are added to the cache.

Note: The Web service must be designed to accept messages that are no older than the messages that have already been removed from the cache. Otherwise, an attacker will be able to replay a message that was previously cleared from the replay cache.

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

Messages cannot be replayed, either accidentally or for malicious intent. Any attempt to replay an intercepted message will result in the message being rejected by the service. Any attempt by the attacker to tamper with the message to spoof the replay mechanism will invalidate the message signature, which causes the service to reject it.

Liabilities

The liabilities associated with the Message Replay Detection pattern include the following:

- The Web service must carefully manage its replay cache to balance scalability and security by clearing the cache at regular intervals.
- If the service is deployed to more than one server in a Web farm, a common replay cache must be used for the cache to be effective. A database is often used for this purpose, but using a database can increase latency of processing messages. The database itself might also be susceptible to denial of service attacks, if an attacker floods connections to the database that is maintained in the Web farm. To help mitigate this issue, you should consider deploying the database server with failover support and ensure that connections to the database server are carefully managed within the Web farm.

Security Considerations

Security considerations associated with the Message Replay Detection pattern include the following:

- The unique identifier for the message must be saved in the replay cache prior to processing the request. This prevents concurrency issues if a second message arrives before the first message has finished executing.
- Some of the steps performed while attempting to detect replayed messages can adversely affect system response time. For example, verifying the signature on the identifier and timestamp is computationally intensive. Reading or updating the replay cache can also impact response time if the cache is on a different computer than the recipient.
- Preventing message replay can help stop a denial of service attack from accessing resources, but it is also possible for an attacker to launch a denial of service attack on the computer that is using message replay detection. The attacker does this by replaying a large number of messages to exploit high resource consumption. To minimize the impact of the attack on system availability and response time, it is important to ensure that the service implements replay detection as efficiently as possible.
- A clock skew value (TTL in seconds) is set on the server to determine the acceptable clock skew between the client and the service. If a message is received outside the acceptable time range, the message will be rejected, even if it is not already present in the cache. Therefore, it is important to ensure that clocks are closely synchronized between the sender and the recipient. This is best achieved by using time synchronization services, with the sender and recipient synchronizing their local clocks to a centralized source. The clock skew must always be less than the time that the messages are held in the cache; if it is not, a replayed message may be accepted because it will already have been deleted from the cache.

- In some cases, a client may not receive a response from a service. As a result, the client will not know whether the request succeeded. A common example that afflicts e-commerce transactions is where a user clicks the **Submit** button twice on a Web form. This scenario requires a different approach, such as the one described in the Idempotent Receiver design pattern. For more information about idempotent Web services, see Idempotent Receiver on the Enterprise Integration Patterns Web site.
- The length of time that messages should be held in the cache varies, depending on the specifics of the recipient. If an application receives a very large number of messages per second, the cache lifetime may be very small, perhaps only a few minutes. In other cases, the cache lifetime may be significantly longer, perhaps hours or days.

Variants

XML signatures provide a basis for effective message identifiers that support message replay detection. They are particularly useful where end-to-end security is required. However, there are alternative ways to implement message replay detection. You can use the following alternatives to XML signatures:

- **Use the full message**. The message itself is unique because the message header contains a timestamp and an XML signature. However, caching the full message can be inefficient because the cache might need to be very large.
- **Use an identifier that is unique to a session, such as a sequence ID**. In this case, each message is assigned a sequence number that is unique within the scope of the active session. This approach requires the client and the server to be synchronized and requires the server to maintain some form of session state to communicate with the client. The session scope may be defined by a span of time that is agreed on by both parties, after which the session must be renewed or renegotiated. Session scope can also be defined by the validity period of a security token used to establish secure communications between the two parties. Both Kerberos and SSL use session-based sequence numbers in their respective replay detection mechanisms.

Related Patterns

Two types of patterns are related to this pattern: a child pattern and a pattern that the Message Replay Detection pattern uses.

The following child pattern is related to the Message Replay Detection pattern:

- **Implementing Message Replay Detection in WSE 3.0**. It provides steps and recommendations to implement message replay detection at the message layer by using WSE 3.0.

The Message Replay Detection pattern uses the following pattern:

- **Data Origin Authentication**. The Data Origin Authentication pattern demonstrates how messages are signed to verify that they are from the intended recipient and have not been altered in transit.

Implementing Message Replay Detection in WSE 3.0

Context

You are implementing a Web service that uses Web Service Enhancements (WSE) 3.0. The Web service accepts messages sent across a public network from clients that manipulate sensitive data or initiate business processes. You need to ensure that the Web service does not process a message that has been intercepted and replayed by an attacker in an attempt to access or manipulate the sensitive data.

Objectives

The objectives of this pattern are to:

- Prevent the service from accepting and processing messages that have expired, while allowing for clock skew.
- Prevent the service from accepting and processing messages that attackers have replayed.
- Support replay attack detection for Web services deployed in a Web farm through a database-supported replay cache.
- Demonstrate an implementation of message replay detection using a WSE 3.0 custom assertion.

Content

This pattern consists of the following sections:

- **Implementation Strategy**. This section provides a high-level description of the strategy used to implement the Message Replay Detection pattern.
- **Implementation Approach**. This section describes the steps required to implement this pattern:
  - Configure the client
  - Configure the service
- **Resulting Context**. This section outlines the benefits, liabilities, and security considerations related to the pattern.

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