Secure Applications Integration with Apache Camel

Sorin POPA
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
sorin.popa.1015@gmail.com

Abstract: Often, applications from a company need to collaborate by exchanging information with each other to support existing business processes. In this paper I present basic concepts about integration applications with Apache Camel and make a demonstration of encrypting messages with Camel components.

Key-Words: Apache Camel, Security, Integration Patterns, Apache Cxf, SSL, TLS

1. Introduction

Enterprise application systems rarely live in isolation. A colossal application that runs all business process is virtually infeasible. Often, applications from a company need to collaborate by exchanging information with each other to support existing business processes.
EAI (Enterprise Application Integration) is the combination of processes, software, standards, and hardware resulting in the seamless integration of two or more enterprise systems allowing them to operate as one. [1]
EAI(Enterprise Application integration) must provide efficient, reliable and secure data exchange between other company's applications or third party applications. Even if different application systems are integrated, they must be able to to evolve without affecting each other. The fact that integrated application systems are independently is an important aspect of EAI because the solution resulted is more tolerant to changes. This is often referred to as Loose Coupling. Loose Coupling can be achieved by utilizing asynchronous communication.
Because integrating enterprise applications is a challenging topic some frequently recurring quandaries and their solution has been described in form of patterns.

2. Problem Formulation

The goal is to explain how to create, using Apache Camel, an environment in which different systems can communicate securely between them using Apache CXF web services.
When securing SOAP messages, various types of threats should be considered. This includes, but is not limited to [2]:

- the message could be modified or read by attackers;
- an attacker could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing.

Confidentiality (protecting the message content from being seen by someone with is not the intended consignee) and integrity (protecting the message content from being altered without detection) are primary security concerns. This objectives can be reach by encrypting and/or digitally signing a body, a header, an attachment, or any commutation of them (or components of them).
Confidentiality assure that information that is consider sensitive or private can't be seen see by erroneous people, while the right people get and can see and the original message. Data encryption is a mundane method of ascertaining confidentiality. Encryption is the most widespread method and it can be found in virtually every major protocol in utilization.
Integrity involves maintaining the consistency, precision, and trustworthiness of data. Data must not be transmuted in transit, and steps must be taken to make sure that data cannot be

ISSN 2067 – 4074

modified by other people without detection.
The SSL/TLS protocol sits between an application protocol layer and a reliable transport layer (such as TCP/IP).
It is independent of the application protocol and can thus be layered underneath many different protocols, for example: HTTP, FTP, SMTP, and so on.
The SSL/TLS protocol supports the following security features [3]:

- *Privacy*—messages are encrypted using a secret symmetric key, making it impossible for eavesdroppers to read messages sent over the connection.
- *Message integrity*—messages are digitally signed, to ensure that they cannot be tampered with.
- *Authentication*—the identity of the target (server program) is authenticated and (optionally) the client as well.
- *Immunity to man-in-the-middle attacks*—because of the way authentication is performed in SSL/TLS, it is impossible for an attacker to interpose itself between a client and a target.

3. Problem Solution
3.1 Apache Camel

Apache Camel is an open source integration framework that make application systems integrating much easier. With Apache Camel, can be defined rules to read, modify and transmit data between application systems. At the core of the Camel framework is a routing engine builder. For create routes, Camel can use a variety of DLSs(Java Domain Specific Language): Java DSL, Spring DSL, Scala DSL, Groovy DSL and others.

3.1.1 Message Model of Apache Camel

Older integration frameworks used two types of objects to integrate application systems. They use application specific object to read information from different systems and then map them into a generic object which was use for routing. One of the major standards of Camel is that it makes no propositions about the sort of information you require to process. This is a significant point, on the grounds that it provides for you, the integration designer, an open door to incorporate any sort of systems, without the need to convert your data to a canonical format.
For representing information Camel use two types of objects: exchange and messages
An exchange in Camel is the message's container during routing. An exchange also provides support for the various types of interactions between systems, also known as message exchange patterns (MEPs). MEPs are used to differentiate between one-way and request-response messaging styles. [4]

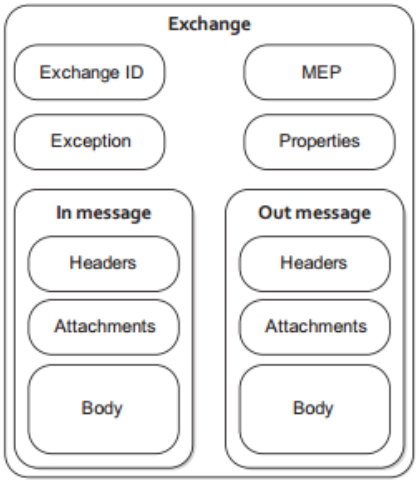

Figure 1. Exchange

- *Exchange ID* - It is used to identify the exchange during routing. The programmer can generate this id, otherwise Camel will do.
- *MEP* — Represent a pattern which will tell if a reply message exist. It can have two values: InOnly and InOut. For InOut a reply message exist and for InOnly a reply message don't exist.
- *Exception* — it is similar to fault tag from SOAP messages. It will be set if an error has been produce during routing.
- *Properties*— it is similar with headers fields from message object. The only thing that differentiate them is the fact that Properties fields exist for the entire duration of exchange object. This field is very useful to hold global level information. Even Camel will add properties to the exchange during routing.

- In message - This object represents the input message in a processor and it is mandatory.
- Out message - If then MEP is set to InOut, out message represent the response message and it will be In message for next processor or endpoint.

Messages are the entities used by systems to communicate with each other when using messaging channels. Messages flow in one direction from a sender to a receiver.
Messages have a body (a payload), headers, and optional attachments, as illustrated in figure 2.

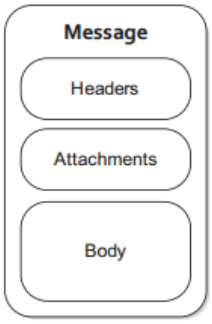

Figure 2. Message

- Headers - Headers are similar to Properties from Exchange. They are name-value pairs; the name is a unique, case-insensitive string, and the value is of type Object. They are useful from holding information regarding sender, encoding of the payload and so on. For example, CXF component put in header name of operation which was call and a programmer can use this information in routing process.
- Attachments - It is an optional field used by typically by web service and email components.
- Body - This object represent information of interest for integrates application system. It's type is `java.lang.Object` so it can store information in any type of objects. Most of the time, sender and receiver use data in different formats so it's the job of integration designer to make sure that systems can understand the message. Components from Apache Camel came to help transforming the data into an acceptable format

Messages also have a fault flag. Some protocols and specifications make a clear distinguish between a valid response and a fault message. Even that both are accepted responses, difference is make by success/failure of operation.
During routing, messages are contained in an exchange.

3.1.2 Architecture of Apache Camel

The Camel Context represent a container which provide access to all services from Camel, most important ones being routes, processors, endpoints, components, a registry, languages and data formats.
In part two of this paper I said that Apache Camel is an routing engine builder. A Camel Context can hold one or more routes which are build using one of Camel domain-specific languages (DSLs). The routing engine use routes from the context as instruction of how messages are routed.
Core of Camel is represented by the processor. In a route, messages are transmitted from a processor to another. Roles of a processor are modifying, using or creating an incoming exchange. Camel provide processor as implementation of enterprise integration pattern but a programmer can easily create custom processor by implementing interface Processor and overwriting function process.
To communicate with different endpoints, camel provide over 80 components. In function of application systems which are present in organization, other component can be created. Component represent endpoint interface and can be accessed using a URI.

3.2 Apache Camel Security

Camel offers several forms and levels of security capabilities that can be utilized on camel routes. These various forms of security may be used in conjunction with each other or separately.
The broad categories offered are [6]:

- Route Security - Authentication and Authorization services to proceed on a route or route segment.
- Payload Security - Data Formats that offer encryption/decryption services at the payload level

ISSN 2067 – 4074

- Endpoint Security - Security offered by components that can be utilized by endpoint Uri associated with the component
- Configuration Security - Security offered by encrypting sensitive information from configuration files

3.2.1 Route Security

Policy driven security capabilities can be used to limit access on routes. Interceptors can be used in Camel processors for controlling authorization and authentication.

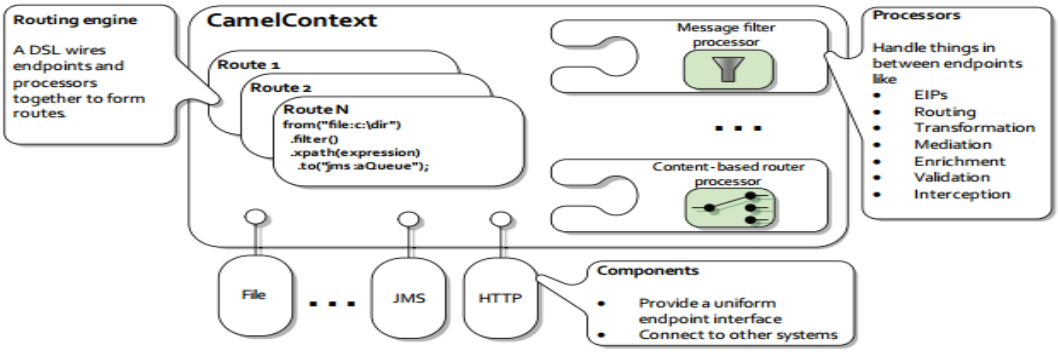

Figure 3. Architecture of Apache Camel (copyright [6])

Apache Camel provides two components for route security:

- Spring Security - can be used to provide a role-based authorization and authentication;
- Shiro Security - an Apache project which it can be use to provide authorization and authentication on a camel route insert a security token into exchange message and manage sessions.

3.2.2 Payload Security

Payload security goal is reach using encryption/decryption services on entire exchange message or on specific sections. Encryption and decryption operations are made using marshal and unmarshal operations. Apache Camel component which provide these services are:

- XMLSecurity DataFormat (XML Encryption support);
- XML Security component (XML Signature support);
- Crypto.

3.2.3 Endpoint Security

Security of endpoints are offered by a limited number of camel component. For this components SSL/TLS can be used to secured messages transmission and for some components (Jetty, Spring Web Services, CXF, JMS), authentication/ authorization capabilities can be provided.

3.2.4 Configuration Security

It's a good practice to keep configurations values, like connection information to database, in properties files. Camel provide component Jasypt for automatic decrypting.

3.3 Securely routing message

In Figure 4 are shown two ways for securing message transmission between two endpoints:

- Endpoint security: In this case both endpoints are secured. The exchange arrive in clear at producer endpoint (on the left) which opens a secure channel to consumer endpoint (on the right). Opening of the secured communication channel is done, typically, using SSL/TLS because all component which can provide endpoint security use SSL. An advantage of endpoint security is the fact that is possible to perform authentication.
- Payload security: In this case both endpoints are insecure. Before then exchange arrive at producer endpoint(on the left) it get encrypted by a processor and another processor decrypt it on the other side CXF component it can be used to expose a web service or to call other web services. On that operation it sends a

SOAP message which is in fact is an XML payload. We can use XMLSecurity Data Format to encrypt/ decrypt XML payload.

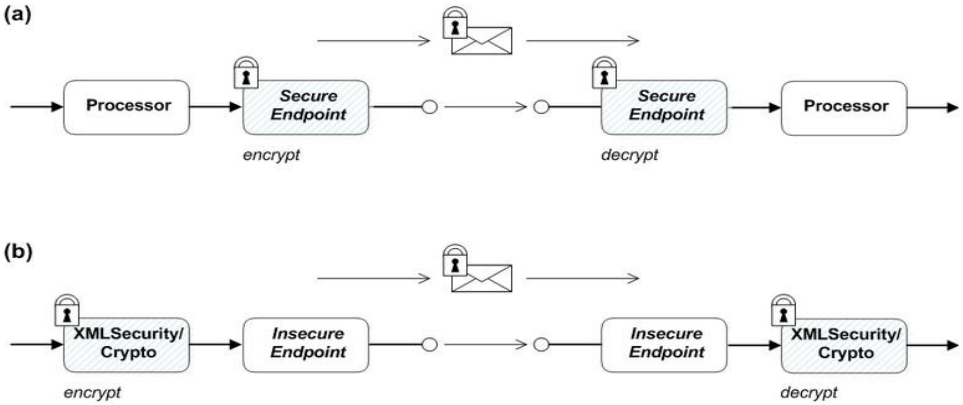

Figure 4. Securing message transmission between two endpoints

In same case, there is no need to encrypt the entire message. XMLSecurity Data Format can encrypt/ decrypt payloads at the Document, Element, and Element Content levels.
Symmetric encryption/decryption is currently supported using Triple-DES and AES (128, 192, and 256) encryption formats. Additional formats can be easily added later as needed. This capability allows Camel users to encrypt/decrypt payloads while being dispatched or received along the route.[6]
An example using symmetric encryption/decryption in Java DSL:

```
String tagXPATH =
"//internetbanking/paymet/account";
boolean secureTagContent = true;
String passPhrase =
mypassphase24bytes";
String algorithm=
XMLCipher.TRIPLEDES;
from("cxf:bean:InternetBankingEndpo
int ")
    .marshal().secureXML(tagXPATH,
secureTagContent, passPhrase,
algorithm)

.unmarshal().secureXML(tagXPATH,
secureTagContent, passPhrase,
algorithm)
```

where:

- tagXPATH is The XPath reference to the XML Element selected for encryption/ decryption
- secureTagContent = true says that content of element is encrypted, not the element;
- passPhrase – pass phase used for encryption
- algorithm for encryption is 3DES

The problem with the above example is than encryption and decryption are done in the same route but often we need to encrypt message on a route and decrypt it on other route. For this operation we must sent symmetric key to consumer. The XMLSecurity Data Format supports asymmetric key encryption. Using this feature we can encrypt symmetric key with consumer public key to ensure than on he can access the "content encryption key". The XMLSecurity Data Format handles all of the logic required to encrypt and decrypt the message content and encryption key(s) using asymmetric key encryption.
Below we have an example using symmetric encryption/decryption and asymmetric encryption/ decryption of the key. The example language is Spring DSL. Symmetric encryption is done using AES 128 CBC and asymmetric encryption is done with RSA.
Sender application is:

```
<camel:keyStoreParameters
id="trustStoreParams"
resource="./sender.ts"
password="password"/>
<camelContext
xmlns="http://camel.apache.org/sche
ma/spring"
xmlns:pay="http://IBK/paymet/">
```

ISSN 2067 – 4074

```
<route>
<from uri="cxf:bean:IBK"/>
<marshal>
<secureXML secureTag="//pay:amount"
secureTagContents="true"
xmlCipherAlgorithm="http://www.w3.o
rg/2001/04/xmlenc#aes128-cbc"
keyCipherAlgorithm="http://www.w3.o
rg/2001/04/xmlenc#rsa-1_5"
recipientKeyAlias="recipient"
keyOrTrustStoreParametersId="trustS
toreParams"/>
</marshal>
```

Receiver application:

```
<camel:keyStoreParameters
id="keyStoreParams"
resource="./recipient.ks"
password="password" />
<camelContext
xmlns="http://camel.apache.org/sche
ma/spring"
xmlns:pay="http:http://IBK/paymet/"
>
<route>
   <from uri="cxf:bean:encrypted"/>
<unmarshal>
<secureXML
secureTag="pay:amount"
secureTagContents="true"
xmlCipherAlgorithm="http://www.w3.o
rg/2001/04/xmlenc#aes128-cbc"
keyCipherAlgorithm="http://www.w3.o
rg/2001/04/xmlenc#rsa-1_5"
recipientKeyAlias="recipient"
keyOrTrustStoreParametersId="keySto
reParams"
keyPassword="privateKeyPassword" />
</unmarshal>
```

4. Conclusion

Integrating application system is not an easy task. An organization can't afford to dismiss old applications and the entire informatics system is growing more and more. Using enterprise integration pattern with Apache Camel we can create a reliable and safe infrastructure ready to integrate other systems.

Acknowledgement

Parts of this paper were presented at The 7th International Conference on Security for Information Technology and Communications (SECITC 2014), Bucharest, Romania, 12-13 June 2014.

References

[1] * * *, EAI.ITtoolbox.com
[2] OASIS Standard - Web Services Security: SOAP Message Security Version 1.1.1
[3]****,http://fusesource.com/docs/broker/5.5 /security/SSL-Intro.html
[4] Claus Ibsen and Jonathan Anstey, Camel in Action, Manning Publications Co,2010, ISBN: 9781935182368
[5] ***, http://fusesource.com/docs
[6] ***, http://camel.apache.org/manual.html