Journal of Mobile, Embedded and Distributed Systems, vol. VI, no. 2, 2014
ISSN 2067 – 4074

Securing Web Services using Service Token Security

Stelian DUMITRA, Bogdan VASILCIUC
Department of Economic Informatics Doctoral School
The Bucharest Academy of Economic Studies
ROMANIA
stelu20d@yahoo.com, bogdanvbg@yahoo.com

Abstract: Web services are distributed components that enable interaction of software components across organizational boundaries. The main advantages of web services are related to the flexibility and versatility: they support a variety of architectures and are independent of platforms and models. Also, they can expose valuable data, applications and systems of organizations to a variety of external threats. Securing web services is one of the most important topics related to them. This paper describes the core web services specifications, the top threats facing web services and the security fundamentals. At the end of the paper is presented a custom authentication and authorization model (brokered authentication) to ensure a robust protection, a model that shows how to authenticate and authorize callers to perform operations and how to access resources. This model uses the following frameworks/standards: Windows Identity Foundation (WIF) to apply the principles of claims-based identity, Windows Communication Foundation (WCF), to develop services/client services and integrate with WIF, and Service Token Security (STS), to issue security tokens.
The conclusions and the future proposed developments are presented in the end of the paper.

Key-Words: web services; windows identity foundation; service token security; windows communication foundation; security assertion markup language; federation; SOAP; WS-Security; collaborative system

1. Introduction

Gradually, the client-server processing model, specific of the first network applications, has been transformed into a model based on service levels.
Considered as the feature of the Internet, the web services are collections of XML/SML standards that allow interaction between systems (programs) and are independent of platform and technology. The role of web services is to share data and to access various services, providing customers a single public interface. They are deployed in Services Oriented Architectures (SOAs), are based on eXtensible Markup Language (XML) and use the Simple Object Access Protocol (SOAP) as protocol to exchange data between applications through dynamic and ad hoc connections. Web services provide a promising framework for development, integration and interoperability of distributed software applications. Wide-scale adoption of the web services technology in critical business applications will depend on the feasibility of building highly dependable services. Web services technology enables interaction of software components across organizational boundaries.
They promise enormous benefits in terms of productivity, efficiency and accuracy. But, while they offer attractive advantages, also web services present daunting challenges regarding the protection and security. Also, they can expose valuable data, applications and systems of organizations to a variety of external threats. These threats are not imaginary. Securing web services is one of the most important topics related to them.
The main advantages of web services are related to the flexibility and versatility: they support a variety of architectures and are independent of platforms and models. Web services are built on multiple technologies that work in conjunction with developing standards to ensure the security and to ensure that they can be combined to work independently of a provider.
A robust solution that ensures the web services protection should have the following basic security requirements: *Authentication, Authorization, Auditing, Confidentiality, Integrity, Non-repudiation and Availability.*

The remainder of the paper is structured as follows. Section 2 provides information about the web services technology and the core web services specifications. Section 3 covers the security fundamentals for web services including the security standards and the top threats security. Section 4 describes a case study based on Security Token Service pattern, Windows Identity Foundation and Windows Communication Foundation for building secure services and on specific web services protocol stack presented in section 2. Section 5 concludes the paper and presents feature proposed developments.

2. Web Services fundamentals

2.1 Web Services

A definition of web service is given by the World Wide Web consortium (W3C):
"A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols" [1].
The role of web services is to share data and to access various services, providing customers a single public interface. They are deployed in Services Oriented Architectures (SOAs), are based on eXtensible Markup Language (XML) and use the Simple Object Access Protocol (SOAP) as protocol to exchange data between applications through dynamic and ad hoc connections. Web services provide a promising framework for development, integration and interoperability of distributed software applications. Because "interoperability" is the key word that defines them, the web services enable applications to work collaboratively. They are successfully used in collaborative systems. An example of such a collaborative system which is based on a multitude of web services is SharePoint. This is a Web Based Collaborative System that facilitates the collaboration within the organization. Using this system the users can create, manage and build collaborative Web sites. It is also accessible through web services for third party programs or for developers.
In essence, web services are characterized by the following:

- Web services can be accessed on the web;
- Web services communicate using XML protocol, a platform-independent and language-neutral;
- Web services can be developed on any programming language and can be deployed on any platform;
- The communication is made using SOAP protocol over HTTP/HTTPS and the messages can be sent over the network in plaintext or encrypted text;
- Web services are registered at Web Service Registries, using UDDI standard, a platform-independent framework as a directory service (repository) where the services can be searched and found;
- The service can be accessed from any type of application client or another service through a prescribed interface.

2.2 Web Services basic standards

In this chapter are analyzed the main standards, technologies and concepts underlying. These standards are used in the web services development and in applications which consume web services.

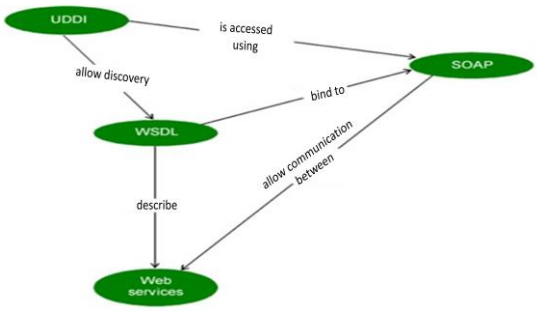

Figure 1. Web services basic standard

The role of the web services specifications is to make the interaction between web service requester and web service provider. In figure bellow are presented the web services basic standards, as well as the connections between them. One of these standards, UDDI, was not yet accepted by certain corporations for

ISSN 2067 – 4074

security reasons and because they are reluctant against the notion of a public registry [1].

2.2.1 SOAP

SOAP, the Simple Object Access Protocol, was originally developed by Microsoft, Developmentor and Userland. Later, IBM and Lotus have contributed to a revised specification, which led to SOAP version 1.1 and published in April 2000. This specification was has been widely accepted throughout the industry and was the basis of several interoperable open source implementations. WS-I .org adopted him as part of his basic profile.
Web services as a SOA component, use the SOAP protocol as a mechanism for the transfer of messages between services described by WSDL interfaces. This concept is illustrated in Figure 2.

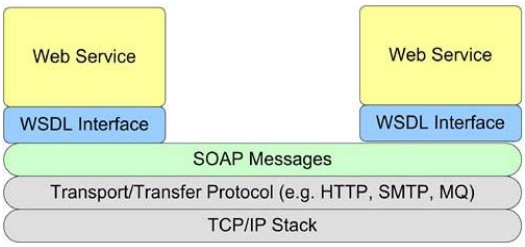

Figure 2. The Web services network

The role of SOAP is to encode the data in XML format and to enable the exchange of XML messages between applications. This uses the request-response model, where the request is given by the *"SOAP client"* and the response is given by the service provider, called *"SOAP server"*. The protocol is used both to send and to receive messages from the Web Service. One advantage is to encapsulate the functionality of the RPC (Remote Procedure Call) using the extensibility and the functionality of the XML [2-3].
A *"SOAP message"* is the basic unit of communication between *"SOAP nodes:* and consists of two elements: the SOAP envelope which is the root of the message (*"SOAP envelope"*) and the *"SOAP body".*
The SOAP envelope includes zero or more SOAP headers. The header contains general information regarding security – authentication and session, and information regarding the processing of the message through intermediate nodes. Authentication data typically are encrypted using the WS-Security standard. The SOAP body never misses and contains information to be transferred between applications. In figure 3 are related two SOAP messages: a request message and a response message.

```
<s:Envelope xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
    <a:Action
s:mustUnderstand="1">http://tempuri.org/ICreditCard/GetCreditCardAmount</a:Action>
    <a:MessageID>urn:uuid:c7e1cd17-cec4-43ba-a203-39d11ae37125</a:MessageID>
    <a:ReplyTo>
      <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
    </a:ReplyTo>
  </s:Header>
  <s:Body>
    <GetCreditCardAmount xmlns="http://tempuri.org/">
 <creditCardNumber>4313786046892009</creditCardNumber>
    </GetCreditCardAmount>
  </s:Body>
</s:Envelope>

<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing">
  <s:Header>
    <a:Action
s:mustUnderstand="1">http://tempuri.org/ICreditCard/GetCreditCardAmountResponse</a:
Action>
    <a:RelatesTo>urn:uuid:e5434dfe-96c5-4398-b6c8-d2e27a50c813</a:RelatesTo>
  </s:Header>
  <s:Body>
    <GetCreditCardAmountResponse xmlns="http://tempuri.org/">
      <GetCreditCardAmountResult>550.00</GetCreditCardAmountResult>
```

```
    </GetCreditCardAmountResponse>
  </s:Body>
</s:Envelope>
```

Figure 3. Simple SOAP messages

The SOAP messages must be sent over secure channels and the protection mechanisms will be described in chapter 3. SOAP was originally an acronym for Simple Object Access Protocol, but since SOAP Version 1.2 (SOAP 1.2 Part 0, 2003; SOAP 1.2 Part 1, 2003) it is technically no longer an acronym.

2.2.2 WSDL – Web Services Description Language

Web Services Description Language, WSDL, is the equivalent of IDL (Interface Description Language), an XML based standard, from CORBA (Common Object Request Broker Architecture) and COM (Component Object Model) and is used to describe services in terms of the supported features, the quality of those functions and the binding mechanisms for interaction with other software agents. WSDL reached version 2.0, but in version 1.1 the D stood for Definition. Version 1.2 of WSDL was renamed WSDL 2.0 because of the major differences between the two versions [1]-[2]-[3].

A WSDL interface consists of two components:

1. *An abstract component* that describes the operations supported by the web service and the types of messages that parameterizes these operations.
2. *A concrete component* that describes how binds the abstract operations to an endpoint and how to map the messages on the transport protocols that the endpoint supports.

In detail, these components use the following elements in the definition of network services:

.

- *Types*: A container for data type definitions using some type system (such as XSD)
- *Message*: An abstract, typed definition of the data being communicated
- *Operation*: An abstract description of an action supported by the service
- *Port Type*: An abstract set of operations supported by one or more endpoints
- *Binding*: A concrete protocol and data format specification for a particular port type
- *Port*: A single endpoint defined as a combination of a binding and a network address
- *Service*: A collection of related endpoints

In Figure 4 is described a WSDL message.

2.2.3 UDDI – Universal Description, Discovery and Integration

UDDI represents a platform-independent framework, based on Extensible Markup Language (XML), a directory service where businesses can register and search for web services. The standard is meant to be open for businesses, enabling them to publish and discover services and to discover the interaction between them over the Internet. UDDI, was not yet accepted by certain corporations for security reasons and because they are reluctant against the notion of a public registry.

ISSN 2067 – 4074

```
- <wsdl:binding name="WSHttpBinding_ICreditCard" type="tns:ICreditCard">
    <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_policy" />
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="GetCreditCardAmount">
      <soap12:operation soapAction="http://tempuri.org/ICreditCard/GetCreditCardAmount" style="document" />
    - <wsdl:input>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCardAmount_Input_policy" />
        <soap12:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCardAmount_output_policy" />
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="GetCreditCardInfo">
      <soap12:operation soapAction="http://tempuri.org/ICreditCard/GetCreditCardInfo" style="document" />
    - <wsdl:input>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCardInfo_Input_policy" />
        <soap12:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCardInfo_output_policy" />
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="GetCreditCards">
      <soap12:operation soapAction="http://tempuri.org/ICreditCard/GetCreditCards" style="document" />
    - <wsdl:input>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCards_Input_policy" />
        <soap12:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <wsp:PolicyReference URI="#WSHttpBinding_ICreditCard_GetCreditCards_output_policy" />
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="CreditCardService">
  - <wsdl:port name="WSHttpBinding_ICreditCard" binding="tns:WSHttpBinding_ICreditCard">
      <soap12:address location="http://idumitra/ccservice/CreditCardService/" />
    - <wsa10:EndpointReference>
        <wsa10:Address>http://idumitra/ccservice/CreditCardService/</wsa10:Address>
      - <Identity xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
          <Dns>idumitra</Dns>
        </Identity>
      </wsa10:EndpointReference>
    </wsdl:port>
  </wsdl:service>
```

Figure 4. Simple WSDL message

A UDDI business registration consists of three components [1-3]:

1. *White pages* - provide basic information about a company, such as the business name, address and contact information. White pages also allow the discovery of services according to unique business identifiers.
2. *Yellow pages* - describe business services using different categorizations—"taxonomies" in UDDI terminology. This information allows for the discovery of business services based on their categorization.
3. *Green pages* - provide technical information on the behaviors and supported functions of services hosted by a business. This includes, for example, the address of the web service, the parameters, etc.

To increase the applications area an extended stack of web services specifications for security, transactions, metadata management, reliability and orchestration has emerged and was standardized. The extended web services architecture stack contains the following specifications [4]:

- *Security*: WS-Authorization, WS-Security Policy, WS-Trust, WS-Secure Conversation, WS Federation, WS-Privacy, XML Encryption, XML Signature
- *Transactions*: WS-Transactions family(WS-Atomic Transaction, WS-

Business Activity, WS Coordination), WS-Composite Application Framework(WS-CAF);
- *Metadata Management*: WS-Addressing, WS-Policy, WS-Metadata Exchange;
- *Reliable Messaging*: WS-Reliability, WS-Reliable Messaging, WS-Eventing, WS Notifications;
- *Orchestration;*
- *Choreography.*

3. Security Fundamentals for Web Services

All software applications, including web services, must satisfy performance requirements, cost, usability and security. Web services security provides the following security services: authentication, authorization, auditing, confidentiality, integrity, non-repudiation and availability. Several organizations, including W3C, OASIS, Liberty Alliance, and various members of the software industry have put together the base of several standards and techniques for web services security. In large part, these standards complement other existing standards, but some may be contradictory or competing. Various XML-based security mechanisms are built around encryption, authentication and authorization mechanisms. The standards are presented bottom-up, from communication layer of the Internet to the application layer. The levels are modeled after the OSI reference model, with the mention that they cannot be interpreted strictly hierarchical [4-5].

- *Secure Sockets Layer (SSL) and Transport Layer Security (TLS)* are security protocols that provide protection of the SOAP messages at the communication level.
- *XML Encryption* and *XML Signature* are the fundamental standards that specify how to encrypt and how to sign the XML data using a variety of supported encryption/signing algorithms.
- *WS-Security and WS-Secure Conversation* defines how to use the XML signature and XML encryption, and security credentials of the SOAP messages (message-level security), providing authentication, integrity and confidentiality of the messages. The WS-Security standard supports tokens for security mechanisms, such as Kerberos tickets and X.509 certificates.
- *WS-Reliability* and *WS-Reliable Messaging* are the fundamental standards regarding the reliability of the messages sent over a public network. They are focused on message delivery guarantee and define the protocols necessary to ensure the non-repudiation of the messages.
- *Security Assertion Markup Language (SAML)* is an XML-based standard for exchanging authorization and authentication data between service provider and service requestor or security domains. SAML solves the problem of Web SSO (single sign-on) by using *"security tokens"* that contains *"assertions"* about subjects for exchanging data between a SAML authority (identity provider) and a SAML consumer (service provider).
- *WS-Policy, WS-Policy Assertion, WS-Policy Attachment and WS-Security Policy*. First standard defines the policy requirements for securing web services. WS-Policy Assertion standard specifies generic security assertions. WS-Policy Attachment and WS-Security Policy offer mechanisms to represent the capabilities and requirements of SOAP messages as policies.
- *eXtensible Access Control Markup Language (XACML) and XACML Profile for Web services* offer security policies and access rights to information for web services and for other resources (digital rights management - DRM).
- *XML Key Management Standard (XKMS)* specifies standard service

ISSN 2067 – 4074

interfaces and protocols for the management of the cryptographic keys.
- *WS-Trust* defines extensions that build on WS-Security to request and issue security tokens and to manage trust relationships.
- *Security Management Standard* defines how to manage credentials by using digital certificates and a PKI (Public Key Infrastructure).
- *Identity Management* defines how to manage users' identity, providing access control to resources.

Although there are a multitude of security standards and technologies available for web services security, they are not always sufficient for an organization or a particular service. For this reason is necessary to understand the treats facing web services. According to WS-I, the top threats facing web services are the following [4]-[5]-[6]:
- *Message alteration*: An attacker can insert, delete or modify information in a message with the purpose to deceive the receiver.
- *Loss of confidentiality*: The information from a message is revealed by an unauthorized person.
- *Falsified messages*: An attacker falsifies the original messages, making the receiver to believe that they come from a trusted sender.
- *Man in the middle:* Occurs when in communication between the sender and receiver appears a third party. The two participants are not aware if the message has not been attacked, allowing attackers to view and / or modify messages.
- *Replay of message*: An attacker send a message already sent previously.
- *Replay of message parts*: The attacker includes portions of one or more previously sent messages in a new message.
- *Denial of Service:* An attacker causes the system to consume resources disproportionately so that valid requests can no longer be met.

The following web services and HTTP standards can provide solutions for protection against these threats: W3C XML Encryption, W3C XML Signature, WS-Security Tokens (user name-password, X.509 certificates, Kerberos tokens, OASIS SAML assertions), W3C WS-Addressing IDs, IETF SSL/TLS protocol, SSL/TLS with client authentication, IETF HTTP Authentication.

4. Securing Web Services using WCF, WIF and STS

The goal of this chapter is to present the implementation of the mechanisms for authentication and authorization of web services using Windows Identity Foundation (WIF), Windows Communication Foundation (WCF) and Service Token Security (STS) technologies.

Windows Communication Foundation (WCF) is Microsoft's unified programming model for building service-oriented applications. WCF provides a common platform for the following technologies: Web Services .ASMX, Web Services Enhancements, .NET Remoting, MSMQ, COM+ and Enterprise Services. The fundamental characteristic of WCF is "*interoperability*". WCF is an excellent framework to create web services and web service clients. The messages can be sent over HTTP SOAP, TCP, named pipes or MSMQ and encoded in text or binary format. WCF enables to build more secured and reliable solutions than ASMX web services [7].

Windows Identity Foundation (WIF) is a set of .NET framework classes for implementing "*claims-based*" applications and, if needed, services that involve tokens issued by a Security Token Service (STS). WIF can be integrated with WCF and together provide authentication and authorization mechanisms for services. WIF is a part of Microsoft's "*Federation Identity*" software family that is an interoperable Identity Metasystem and contains other three components: "*Active Directory Federation Services (ADFS) 2.0*" **(**previously known as "*Geneva" Server***),**

"*Windows Azure Access Control Services (ACS) and Windows CardSpace 2.0"***.** Summary, WIF presents the following benefits [8-9]:

- Building claims-aware applications (relying party applications). It offers a claims model and the user's authorization policies are based on claims (the roles and/or permissions are replaced with claims).
- Decoupling applications and services from the authentication mechanism.
- Building custom STSs - authentication services that issue tokens.
- Enabling federation between two or more partners – when a user is authenticated in domain, he can access the resources in another domain if between domain's STS are established a trust.
- Supporting identity delegation scenarios, so it helps to maintain the identities across the service boundaries and enable the single sign-on scenarios.

Any of the scenarios presented above can be based on *"active federation"* (Windows client-based) or *"passive federation"* (browser-based)[8]. The active federation scenarios are based on the WS-Federation Active Requestor Profile and the WS-Trust specification. The WS-Trust specification defines a contract with four service operations: Issue, Validate, Renew and Cancel, operations that are called by clients to request a security token, to validate a security token, to renew an expired security token or to cancel a security token. When a client call a service operation, it send a message to STS in the form of Request Security Token (RST) and STS send back to the client a message in the form of Request Security Token Response (RSTS) according to the WS-Trust specification. These issued tokens can contain the claims and can be SAML 1.1 or SAML 2.0 token [10]. In the active federation models the client (like a WCF client) can parse the policy and implement WS-Trust directly. In the passive federation models the clients (like web-browsers) can't parse the tokens directly (in browsers a token is stored in a cookie), because they have limitations, like to perform cryptographic operations other than SSL activities.

In figure 7 is presented a simple active federation scenario in order to support the previous presented. The scenario consists from a WPF client, an IP (STS) web service and a WCF service. The WCF service, "*CreditCardService"*, exposes the contract "*IcreditCard"*, as shown in figure 5. The WCF client calls the operations service using a WCF proxy and is configured to use custom authentication based on claims.

```
[ServiceContract(Namespace =
"http://schemas.microsoft.com/ws/2008/0
9")]
public interface ICreditCard
{
[OperationContract]
decimal GetCreditCardAmount(string
creditCardNumber);
[OperationContract]
CreditCardInfo GetCreditCardInfo(string
creditCardNumber);
[OperationContract]
void UpdateAmountCreditCardInfo(string
creditCardNumber, decimal amount);
[OperationContract]
IEnumerable<CreditCardInfo>
GetCreditCards();
}
```

Figure 5. CreditCardService contract

Also, the claims-based WCF service, CreditCardService, is configured to expose federation endpoints. This service receives issued tokens by the trusted service, STS (such as SAML tokens), validates them and performs the subject's authorization. WS2007FederationHttpBinding is the latest binding that supports federated security and WS-Trust1.3.

```
<microsoft.identityModel>
  <issuerNameRegistry type="Credits.TrustedIssuerNameRegistry, Credits"/>
  <securityTokenHandlers>
    <remove type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
Microsoft.IdentityModel, Version=0.5.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
    <add type="Microsoft.IdentityModel.Tokens.Saml11.Saml11SecurityTokenHandler,
Microsoft.IdentityModel, Version=0.5.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
```

ISSN 2067 – 4074

```
      <samlSecurityTokenRequirement audienceUriMode="Never">
        <nameClaimType value="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"/>
        <roleClaimTypes>
          <add value="http://schemas.microsoft.com/ws/2008/09/identity/claims/permission"/>
        </roleClaimTypes>
      </samlSecurityTokenRequirement>
    </add>
  </securityTokenHandlers>
      <serviceCertificate>
        <certificateReference findValue="CN=STSCredit" storeLocation="LocalMachine"
storeName="My" x509FindType="FindBySubjectDistinguishedName"/>
      </serviceCertificate>
</microsoft.identityModel>
<system.serviceModel>
  <services>
    <service name="Credits.CreditCardService" behaviorConfiguration="federationBehavior">
      <endpoint binding="wsFederationHttpBinding"
bindingConfiguration="wsFederationConfiguration" contract="Credits.ICreditCard"/>
      <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
    </service>
  </services>
  <bindings>
    <wsFederationHttpBinding>
      <binding name="wsFederationConfiguration">
        <security mode="Message">
          <message issuedKeyType="SymmetricKey" issuedTokenType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-.1#SAMLV1.1" negotiateServiceCredential="true">
            <claimTypeRequirements>
              <add claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"
isOptional="false"/>
              <add
claimType="http://schemas.microsoft.com/ws/2008/09/identity/claims/permission"
isOptional="false"/>
            </claimTypeRequirements>
            <issuer address="http://sdumitra:8080/STSService/STS.svc"/>
            <issuerMetadata address="http://sdumitra:8080/STSService/STS.svc/mex"/>
          </message>
        </security>
      </binding>
    </wsFederationHttpBinding>
  </bindings>
</system.serviceModel>
```

Figure 6. Configuring CreditCardService

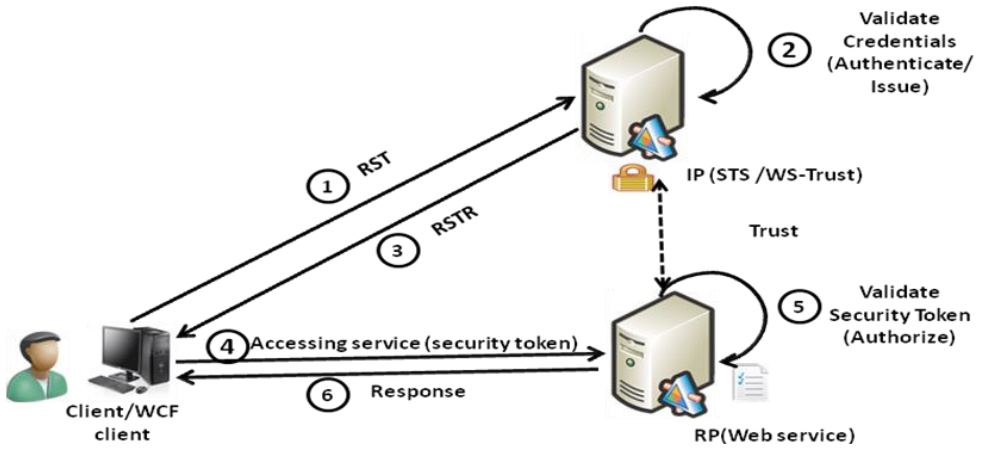

Figure 7. Service Token Security (STS) pattern

The claims-based architecture presented above defines the following actors [8-9]:

- *Subject (client):* the entity that wants to authenticate and access the web service. The entity can be a simple user or an application code/client (WCF client).
- *Relaying Party (RP):* is the web service called by subject. The RP can be also a

web application, can consume the tokens provided by IP (STS), extracts the claims from tokens and authorize the subject/caller to perform operations.

- *Identity Provider (IP) or STS (Service Token Security):* is implemented as a web service and holds the authentication. STS builds, signs and issues security tokens with claims identifying the subject/caller, according to the WS-Trust and WS-Federation protocols. Also, STS validated the credentials presented by caller and authenticate it.
- *Identity:* describes a set of attributes that describe (claims) an entity.
- *Claim:* contains information about authenticated entity such as: name, email, date of birth, department, age, role, permission etc. When STS performs the caller authentication, it issues a security token that contains a set of claims. Based on these claims the RP take the decision to authorize the subject to call the RP's operations.
- *Security Token (RST/RSTR):* is a serialized set of claims that is digitally signed or not by the issuing authority. The security token are issued by STS and is carried in the security header of the SOAP envelope. A security token can be a username/password combination, a simple string, a SAML token in XML format or binary-based such as X.509 certificates and Kerberos tickets. To secure a token, STS can apply a series of cryptographic operations.
- *Issuing Authority:* says STS's how to issues security tokens, such as domain controllers that issues Kerberos tickets or certificate authorities that issues X.509 certificates.

The authentication and authorization process using claims-based authorization approach and presented in figure above follows the steps[11]-[12]:

1. *The client initializes and sends the authentication request to STS.* The request authentication sent to STS contains a Request Security Token (RST) message. The RST contains the client credentials that can be send directly in plain/cipher mode or can be send using a token issued by an authentication broker, such as Kerberos token or X.509 certificate. Is very important that RST and RSTR messages to be secured. In this case the credentials will be sent directly in encipher mode.
2. *STS receives the RST token from client and validates the credentials.* If the credentials were validated, the STS issues a security token as a SAML assertion and applies the policies rules as claims. The token is then signed with a X.509 certificate. CreditCardService can read the token because between STS and WCF service exist a trusted relationship, so the WCF service must recognize the certificate that signed token. The user information are stored in a database, but they can be stored in local memory, encrypted files, ADFS 2.0 that authenticates users against the Windows domain an issues claims according ADFS configuration. The policies applied by STS as claims can be stored in local memory, database etc.
   To validate the credentials presented by client STS must implement a specific class by credentials type. In this case STS implements serNameSecurityTokenHandler class.

ISSN 2067 – 4074

```
public class CustomUserNameSecurityTokenHandler : UserNameSecurityTokenHandler
{
    public CustomUserNameSecurityTokenHandler()
    {
    }
    public override bool CanValidateToken
    {
        get { return true; }
    }
    public override ClaimsIdentityCollection ValidateToken(SecurityToken token)
    {
        UserNameSecurityToken userNameToken = token as UserNameSecurityToken;
        if (userNameToken == null) throw new SecurityException(string.Format("Invalid token provided: {0}.", token.GetType()));
        var user = DBStore.ReadUser(userNameToken.UserName);
        if (user != null && user.UserName == userNameToken.UserName && userNameToken.Password.Equals(user.Password))
        {
            ClaimsIdentityCollection identities = new ClaimsIdentityCollection();
            ClaimsIdentity claimsIdentity = new ClaimsIdentity("CustomUserNameSecurityTokenHandler");
            claimsIdentity.Claims.Add(new Claim(ClaimTypes.Name, userNameToken.UserName));
            identities.Add(claimsIdentity);
            return identities;
        }
        else
            throw new SecurityException(string.Format("Invalid user name/password: {0}", userNameToken.UserName));
    }
}
```

Figure 8. Validating security token

```
<microsoft.identityModel>
        <securityTokenHandlers>
                <remove
type="Microsoft.IdentityModel.Tokens.WindowsUserNameSecurityTokenHandler,
Microsoft.IdentityModel, Version=0.5.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
                <add type="STS.CustomUserNameSecurityTokenHandler, STS"/>
        </securityTokenHandlers>
</microsoft.identityModel>
```

Figure 9. Configuring STS to use CustomUserNameSecurityTokenHandler

3. *STS send the security token to client.* The security token issued at previous point will be sent to client as SAML token in a RSTR message. RSTR contains a set of three claims: *username, read permission and update permission*. These claims will be used by WCF service to authorize the client.

```
ClaimTypes.Name = http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
public const string Permission =
http://schemas.microsoft.com/ws/2008/09/identity/claims/permission
public const string Read =
http://schemas.microsoft.com/ws/2008/09/identity/claims/permission/read
public const string Update = http://schemas.microsoft.com/ws/2008/09/identity/claims/permission/update

protected override IClaimsIdentity GetOutputClaimsIdentity(IClaimsPrincipal principal,
RequestSecurityToken request, Scope scope)
{
            IClaimsIdentity claimsIdentity = new ClaimsIdentity();

            claimsIdentity.Claims.Add(new Claim(ClaimTypes.Name, principal.Identity.Name));
            claimsIdentity.Claims.Add(new Claim(Constants.ClaimTypes.Permission,
Constants.Permissions.Read));
            claimsIdentity.Claims.Add(new Claim(Constants.ClaimTypes.Permission,
Constants.Permissions.Update));

            return claimsIdentity;
 }
```

Figure 10. Applying the client's policies and issuing the security token

The GetOutputClaimsIdentity method defines the set of claims that will be serialized into a security token.

4. *The client initializes and sends a request message to the WCF service.* After the client receives the security token from the STS, will initialize and send a request message to the WCF service, for example he will try to call an operation exposed by the WCF contract service.
   - The client proxy class must authenticate the client to the STS with

a user name and a password using message security.

```
<ws2007HttpBinding>
    <binding name="stsBinding" >
        <security mode="Message">
            <message clientCredentialType="UserName"
                     negotiateServiceCredential="false"
                     algorithmSuite="Default"
                     establishSecurityContext="false" />
        </security>
    </binding>
</ws2007HttpBinding>

CreditCardClient.CreditCardClient client = new CreditCardClient.CreditCardClient();
client.ClientCredentials.UserName.UserName = "stelian.dumitra";
client.ClientCredentials.UserName.Password = "password";
client.GetCreditCards();
```

Figure 11. Client's request to service

5. *The WCF service validates the security token presented by client and performs the authorization.* The service verifies that the token was issued by STS and was not modified after it was released (verifies the integrity – digital signature). The service uses the STS public certificate included in the request. Then the security token will be decrypted and after checking the set of claims the service will take decisions on the client's authorization. The verification is made by the "*PrincipalPermission attribute"*. This attribute performs declarative authorization.

```
[PrincipalPermission(SecurityAction.Demand, Role = Constants.Permissions.Update)]
public void UpdateAmountCreditCardInfo(string creditCardNumber, decimal amount)
{
    var creditCards = this.GetCreditCards();
    var creditCardInfo = creditCards.Where(c =>
c.CardNumber.Equals(creditCardNumber)).FirstOrDefault();
    if (creditCardInfo != null)
    {
        if (amount > creditCardInfo.TotalUsed)
            throw new SecurityException("The amount must be lower that total used amount!");
        creditCardInfo.TotalUsed -= amount;
       DBStore.UpdateCreditCardInfo(creditCardInfo)
    }
}

[PrincipalPermission(SecurityAction.Demand, Role = Constants.Permissions.Read)]
public CreditCardInfo GetCreditCardInfo(string creditCardNumber)
{
    var creditCards = this.GetCreditCards();
    return creditCards.Where(c => c.CardNumber.Equals(creditCardNumber)).FirstOrDefault();
}
```

Figure 12. Implementing the CreditCardService

6. *Service initializes and sends the response to the client as a message (optionally).* After WCF service authorizes the client, it will send an optional message to client.

5. Conclusion

The great advantage of Web Services is that they can integrate different platforms and allow the interoperability between different distributed systems. But, while they offer attractive advantages, also web

ISSN 2067 – 4074

services present daunting challenges regarding the protection and security. Also, they can expose valuable data, applications and systems of organizations to a variety of external threats. Securing web services is one of the most important topics related to them. Web services are built on multiple technologies that work in conjunction with developing standards to ensure the security and to ensure that they can be combined to work independently of a provider.

The proposed model for securing web services using STS has the following advantages:

- Decoupling applications and services from the authentication mechanism. The model consists from three parties: the client which emits requests to web service, but it must authenticate via STS; the STS, a web services that validates credentials, authenticates the client and issues the SAML security tokens with set of claims to call the operations exposed by web service; and RP (web service), that will receive the security token from client and use the claims to authorize the caller to perform operations and access resources.
- Building claims-aware applications (relying party applications). It offers a claims model and the user's authorization policies are based on claims (the roles and/or permissions are replaced with claims).
- Building custom STSs - authentication services that issue tokens.
- The solution is not dependent by other mechanisms, such as the Kerberos protocol or certificates X.509 for securing messages.

As a general conclusion, securing web services against threats is very importantly because they can expose valuable data. Using a brokered authentication model based on claims and STS will decrease the security risks.

Acknowledgment

Parts of this research have been published in the Proceedings of the 6th International Conference on Security for Information Technology and Communications, SECITC 2013.

References

[1] Web of Services**,** [Online] Available: *http://www.w3.org/standards/webofservices*

[2] Tzima, F. and Mitkas, P., Web Services Techology, *Information Science Reference*, pp. 25-44, 2008

[3] Dumitrache, M., Dumitra, S., Baciu, M., Web Services Integration with Distributed Applications, *Journal of Applied Quantitative Methods*, vol. 5, no. 2, 2010, pp 223-233

[4] Bertino, E., Martino, L., Paci, F., Squicciarini, A. Security for Web Services and Service-Oriented Architectures, *Springer-Verlag Berlin Heidelberg*, 2010, pp 1-77

[5] Maier, J.D., Farre, C., Taylor, J., Improving Web Services Security - Scenarios and Implementation Guidance for WCF**,** *Microsoft*, 2009, [Online] Available: *http://wcfsecurity.codeplex.com/*

[6] Singhal, A., Winograd, T., Scarfone, K., Guide to Secure Web Services – Recommendations of the National Institute of Standards and Techonology, NIST Special Publications, August 2007, [Online] Available http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf

[7] Cheng, S., Microsoft Windows Communication Foundation 4.0 Cookbook for Developing SOA Application, Packt Publishing, 2010

[8] Bertoccio, V., Programming Windows Identity Foundation, Microsoft Press, Redmond Washington, 2011

[9] Brown, K., Mani, S., Microsoft Windows Identity Foundation (WIF) Whitepaper for Developers, Microsoft Corporation, 2009

[10] SAML, [Online] Available: *http://saml.xml.org/*

[11] Michèle Leroux Bustamante, Project, Claims-Based Authorization with WIF, [Online] Available: H*ttp://msdn.microsoft.com/en-us/magazine/ee335707.aspx*

[12] Windows Identity Foundation, [Online] Available: *http://msdn.microsoft.com/en-us/li\brary/hh377151.aspx]*