

## Technologie internetowe laboratorium nr 5

### Zabezpieczanie usług sieciowych z wykorzystaniem HTTP Basic Authentication oraz HTTPS

Paweł Czarnul, KASK, WETI, Politechnika Gdańska  
pczarnul@eti.pg.gda.pl

#### 1. Wprowadzenie

Celem niniejszego laboratorium jest zabezpieczenie usług sieciowych dostępnych na serwerze za pomocą mechanizmów HTTP Basic Authentication oraz HTTPS ([2], [1]). Wywołanie metody dostępnej w ramach usługi sieciowej zainstalowanej na serwerze np. Tomcat/AXIS jak pokazano w ćwiczeniu 4, następuje z wykorzystaniem protokołu SOAP, w którym zakodowane są argumenty wejściowe jak i wyniki działania metody. Do przekazania danych pomiędzy klientem a serwerem wykorzystywany jest protokół HTTP. Z punktu widzenia zabezpieczenia usługi rodzi to następujące problemy:

1. Klient nie jest identyfikowany, wywołania metody tej samej usługi przez np. różne osoby nie są rozróżniane. Stąd niemożliwe staje się np. naliczanie opłat za wywołanie metody.
2. Argumenty wejściowe i wyjściowe metody przekazywane są z wykorzystaniem protokołu HTTP, nie są więc szyfrowane co potencjalnie umożliwia podsłuchanie komunikacji pomiędzy klientem a serwerem.

W ramach przykładów, które rozwiążą ww. kwestie, wykorzystany zostanie serwer Tomcat/AXIS, który pozwoli na zainstalowanie usługi sieciowej napisanej w Javie.

Podstawowa konfiguracja serwera Tomcat/AXIS jak również kod klienta wywołującego metodę usługi sieciowej i jego kompilacja i uruchomienie objęte są instrukcją ćwiczenia nr 4.

#### 2. Wykorzystanie mechanizmu HTTP Basic Authentication

Mechanizm ten umożliwia wymuszenie zalogowania się przez klienta na serwerze poprzez podanie loginu użytkownika oraz hasła. Niestety, login i hasła nie są szyfrowane w sposób uniemożliwiający ich odczytanie. Mechanizm ten pozwala jednak na identyfikację użytkownika przy założeniu, iż różni użytkownicy dysponują różnymi identyfikatorami użytkownika w postaci ciągu znaków login.

Ww. sposób wymaga konfiguracji serwera w postaci określenia użytkowników, którzy mogą się do systemu logować, a w szczególności:

1. Login, hasło oraz rolę użytkownika w postaci linii w pliku jakarta-tomcat-4.1.18/conf/tomcat-users.xml:

```

<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="runtaskuser"/>
  <role rolename="onjavauser"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="runtaskuser" password="uio9" roles="runtaskuser"/>
  <user username="bob" password="password" roles="onjavauser"/>
</tomcat-users>

```

Zdefiniowano użytkownika `runtaskuser`, określono hasło oraz rolę, do której referencja występuje z kolei w pliku katalogu systemu AXIS tj.:

`jakarta-tomcat-4.1.18/webapps/axis/WEB-INF/web.xml`

W pliku tym, na końcu elementu `</web-app>` należy zdefiniować URL, który obejmować będzie rola określona powyżej poprzez umieszczenie np.:

```

<security-constraint>
  <web-resource-collection>
    <web-resource-name>RunTaskApplication</web-resource-name>
    <url-pattern>*/</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>runtaskuser</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>RunTask Application</realm-name>
</login-config>

```

`login-config` określa metodę identyfikacji użytkownika, `web-resource-collection` określa zbiór adresów URL (w tym przypadku wszystkie strony w danym katalogu systemu AXIS), `auth-constraint` określa rolę, dla której zezwala się na dostęp do podanego URL.

Należy zauważyć, iż kod metod usługi sieciowej nie odwołuje się do roli, która ma dostęp do usługi, uprawnienia są więc w tym przypadku elementem konfiguracji serwera.

Kod usługi sieciowej, wykorzystywany w niniejszej instrukcji, umożliwia uruchomienie na serwerze polecenia podanego jako argument:

```
import java.io.*;
```

```

public class RunTaskServer {
    public int RunTask(String taskname)
    {

        try {
            Process p = Runtime.getRuntime().exec(taskname);
        } catch (IOException e1) {
            System.err.println(e1);
            System.exit(1);
        }
        return 0;
    }
}

```

Kod klienta wywołujący metodę podaną powyżej i zainstalowaną jako plik JWS (Java Web Service) w katalogu jakarta-tomcat-4.1.18/webapps/axis/RunTaskServer.jws, uzupełniony jest o zdefiniowanie loginu i przekazanie hasła w celu uwierzytelnienia na serwerze:

```

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.rpc.ParameterMode;

public class RunTaskClientBASIC
{
    public static void main(String [] args) throws Exception {

        Options options = new Options(args);

        String endpoint = "http://localhost:" + options.getPort() +
            "/axis/RunTaskServer.jws";

        args = options.getRemainingArgs();

        String method = "RunTask";

        String s1 = new String(args[0]);

        Service service = new Service();

        Call call = (Call) service.createCall();

        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        call.setOperationName( method );
        call.addParameter( "op1", XMLType.XSD_STRING, ParameterMode.IN );

```

```

call.setReturnType( XMLType.XSD_INT );

call.setUsername("runtaskuser");
call.setPassword("uio9");

Integer ret=(Integer) call.invoke( new Object [] { s1 });

System.out.println("Got result : " + ret);

}
}

```

### 3. Wykorzystanie HTTPS do wywołania metody usługi sieciowej

Protokół HTTPS (HTTP+SSL) pozwala na szyfrowanie komunikacji pomiędzy klientem a serwerem, nie zachodzi więc obawa o podejrzenie przesyłanych danych przez osoby postronne. Protokół HTTPS wykorzystuje techniki kryptografii asymetrycznej i symetrycznej.

Konfiguracja serwera Tomcat do obsługi HTTPS sprowadza się do odkomentowania odpowiednich deklaracji w pliku konfiguracyjnym jakarta-tomcat-4.1.18/conf/server.xml tj. w wersji 4.1.18:

```

<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="true"
  acceptCount="100" debug="0" scheme="https" secure="true"
  useURIVValidationHack="false" disableUploadTimeout="true">
  <Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
    clientAuth="false" protocol="TLS" />
</Connector>

```

Tomcat obsługiwać będzie odtąd zgłoszenia na porcie 8080 oraz 8443 (domyślnie), ostatnie z wykorzystaniem HTTPS.

Konfiguracja klienta usługi sieciowej, która ma być wywołana z wykorzystaniem HTTPS sprowadza się do ustawienia odpowiedniego adresu URL (HTTPS zamiast HTTP) oraz wskazania odpowiedniego pliku keystore (certyfikat SSL), który umożliwi szyfrowaną komunikację pomiędzy klientem a serwerem. Wygenerowanie pliku (jednego w przypadku klienta i serwera na tej samej maszynie) sprowadza się do wywołania:

```
keytool -genkey -alias tomcat -keyalg RSA
```

Dodatkowe elementy kodu klienta, które należy umieścić przed wywołaniem metody usługi sieciowej to:

```
String endpoint =  
"https://localhost:8443/axis/accessiblebyHTTPS/RunTaskServer.jws";  
  
System.setProperty("javax.net.ssl.trustStore",  
    "/home/pczarnul/.keystore");
```

#### 4. Połączenie metod HTTP Basic Authentication oraz szyfrowanej transmisji HTTPS

Wyżej wymienione techniki mogą zostać połączone ze sobą w celu zapewnienia identyfikacji użytkownika oraz szyfrowania transmisji danych. Należy przeprowadzić kroki konfiguracji serwera dla obu wymienionych metod oraz zmodyfikować kod klienta z elementami pokazanymi wyżej dla obu wymienionych metod.

Pomimo tego, iż użytkownik został zidentyfikowany na podstawie loginu i hasła oraz stosowana jest komunikacja szyfrowana, serwer nie może mieć absolutnej pewności co do tożsamości klienta tj. nie wie, kim faktycznie jest klient o danym loginie i hasle. W praktyce sklepy internetowe często nie rozwiązują tego problemu zakładając, iż użytkownik zakładając konto w sklepie o danym loginie i hasle podaje również poprawne dane osobowe oraz adres, pod który wysyłany jest następnie towar np. za zaliczeniem pocztowym. Z reguły, używana jest szyfrowana transmisja HTTPS, zaś sama identyfikacja użytkownika następuje na poziomie aplikacji poprzez przekazanie loginu i hasła np. przez pola formularza.

Java posiada odpowiednie API, które pozwala na implementację podpisu cyfrowego danych oraz weryfikację podpisu. Klient generuje klucze prywatny i publiczny, następnie podpisuje dane z użyciem klucza prywatnego, przekazuje dane wraz z podpisem i kluczem publicznym do weryfikacji serwerowi, który jest w stanie stwierdzić, czy podpis faktycznie odnosi się do danych, które zostały przekazane. Serwer musi mieć pewność, iż klucz publiczny należy do określonego klienta. Certyfikat klienta zawierać będzie klucz publiczny. Certyfikat będzie podpisany przez jednostkę certyfikującą, która potwierdza, iż klucz publiczny należy do określonego klienta. API oraz przykłady Javy odnośnie podpisów cyfrowych można znaleźć w [3] i [4].

Przykładowy klient wykorzystujący HTTP Basic Authentication oraz HTTPS ma następującą postać:

```
import org.apache.axis.client.Call;  
import org.apache.axis.client.Service;  
import org.apache.axis.encoding.XMLType;  
import org.apache.axis.utils.Options;  
  
import javax.xml.rpc.ParameterMode;  
  
public class RunTaskClientHTTPSBASIC  
{
```

```
public static void main(String [] args) throws Exception {
```

```
Options options = new Options(args);
```

```
String endpoint =  
"https://localhost:8443/axis/accessiblebyHTTPS/RunTaskServer.jws";
```

```
System.setProperty("javax.net.ssl.trustStore",  
"/home/pczarnul/.keystore");
```

```
args = options.getRemainingArgs();
```

```
String method = "RunTask";
```

```
String s1 = new String(args[0]);
```

```
Service service = new Service();
```

```
Call call = (Call) service.createCall();
```

```
call.setTargetEndpointAddress( new java.net.URL(endpoint) );  
call.setOperationName( method );  
call.addParameter( "op1", XMLType.XSD_STRING, ParameterMode.IN );  
call.setReturnType( XMLType.XSD_INT );
```

```
call.setUsername("runtaskuser");  
call.setPassword("uio9");
```

```
Integer ret=(Integer) call.invoke( new Object [] { s1 } );
```

```
System.out.println("Got result : " + ret);
```

```
}  
}
```

## Literatura

1. AXIS User's Guide. <http://ws.apache.org/axis/java/user-guide.html>
2. Gopalakrishnan U, Rajesh Kumar Ravi, Web services security, Part I, <http://www-106.ibm.com/developerworks/webservices/library/ws-sec1.html>
3. Thomas Gutschmidt Introduction to Digital Signatures in Java, <http://www.developer.com/java/other/article.php/630851>
4. The Java™ Tutorial, Trail: Security in Java 2 SDK 1.2, Lesson: Generating and Verifying Signatures, <http://java.sun.com/docs/books/tutorial/security1.2/apisign/index.html>