Guião de demonstração de segurança

Sistemas distribuídos

3ª Entrega

Grupo 67

André Nunes nº 64728
André Vieira nº 79591
Ricardo Marques nº 81778

Instruções de instalação

- Obter o projeto através do repositório

```
git clone https://github.com/tecnico-distsys/A67-SD18Proj.git
```

- Abrir o terminal na pasta do projeto e gerar os ficheiros necessários:

```
mvn generate-sources
```

- Executar na pasta ws-handlers:

```
mvn clean install
```

- Executar Station-ws, binas-ws, e binas-ws-cli por esta ordem, ao executar nas respetivas pastas o seguinte comando:

```
mvn compile exec:java
```

Demonstração de Segurança

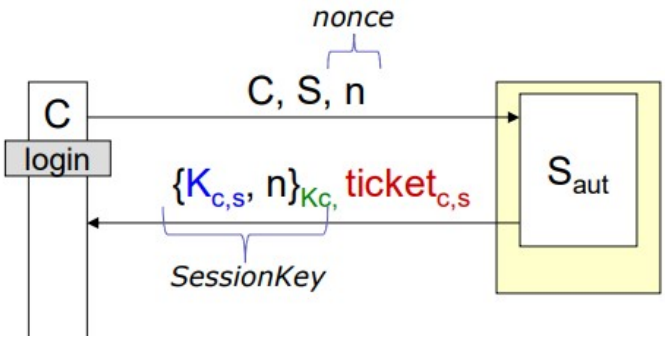
Caso F1: passos para demonstrar o funcionamento normal da segurança

Como ponto de partida da demonstração do projeto, introduzimos na função main do ficheiro BinasClientApp do binas-ws-cli, as seguintes linha de código:

```
System.out.println("Activating User...");
client.activateUser( email: "alice@A67.binas.org");

System.out.println("Renting Bina...");
client.rentBina( stationId: "A67_Station1",  email: "alice@A67.binas.org");
```

onde é feito um pedido para ativar o utilizador através de *activateUser,* fazendo de seguida um pedido para alugar uma bicicleta através de *rentBina.*
Ao executar o pedido para ativar o utilizador, a mensagem SOAP que sai do cliente é intercetada pelo handler **KerberosClientHandler,** que irá contactar o servidor Kerby para se autenticar e obter a chave de sessão e um ticket, de acordo com o seguinte esquema:



É enviado o id do cliente (email) representado por C, a identificação do servidor (S) e um nounce gerado aleatóriamente, que irá ser usado para detectar ataques de replay.
Ao serem validados, o Sauth devolve uma SessionKey que o handler irá decifrar com a chave do cliente (gerada com base na sua password), e também um ticket cifrada com a chave do servidor. Aqui, vai ser obtido o nounce devolvido pelo servidor ao decifrar a sessionKey com a chave do cliente, e vai ser possível comparar com o nounce enviado no pedido, sendo este o primeiro mecanismo de segurança implementado, garantido assim que a resposta é a esperada de acordo com o pedido ao verificar que o nounce recebido é o mesmo que foi enviado, salvaguardando assim possíveis ataques.
De seguida, o handler do cliente irá criar um autenticador, que irá conter o id do cliente (email) e um timestamp com a data em que foi criado, sendo de seguida cifrado com a chave de sessão.
O handler do cliente irá então colocar o ticket e o autenticado no header da mensagem soap. Podemos verificar que estas operações se realizam com sucesso através das mensagens de output no terminal do cliente:

```
Creating client using UDDI at http://a67:usxLuEy@uddi.sd.rnl.tecnico.ulisboa.pt:9090 for server with name A67_Binas
Activating User...
Message destination: http://localhost:8080/binas-ws/endpoint
Client Handler: Nounce verificado com sucesso
Client Handler: Ticket adicionado ao SOAP Header
Client Handler: Auth adicionado ao SOAP Header
```

A mensagem segue então para o servidor, sendo primeiro intercetada pelos handlers de segurança:
BinasAuthorizationHandler: Verifica que o email que segue no body da mensagem SOAP corresponde ao user autenticado, evitando que um atacante se faça passar por outro utilizador alterado esse campo na mensagem.
MACHandler: Garante a integridade do pedido, ao concatenar a o body da mensagem com a chave de sessão e gerar um resumo, onde o cliente irá colocar esse resumo no header da mensagem soap que será comparada pelo servidor, que irá repetir os passos e verificar se os resumos são iguais. Se forem, garante-se que a o corpo da mensagem não foi modificado.
KerberosServerHandler: Irá avaliar a frescura do pedido, ao aceder ao autenticador no header da mensagem, decifrá-la com a chave de sessão, e verificar se a data se encontra dentro dos limites T1 e T2, garantido assim que não se trata de uma mensagem antiga. Coloca de seguida Treq que corresponde à data recebida no header da mensagem para que seja validada pelo cliente, garantido assim que a mensagem corresponde ao pedido efetuado.
O sucesso de execução destas operações pode mais uma vez ser verificado no terminal do servidor através das mensagens de output:

```
Server handler: Timestamp validado com sucesso
MAC Handler: Teste de integridade validado com sucesso. MAC iguais
BinasAuthHandler: Email do pedido corresponde ao email autenticado
MAC Handler: Digest adicionado ao header da mensagem
Server handler: Treq adicionado ao SOAP Header
```

Por fim, o handler do cliente irá interceptar a mensagem devolvida pelo servidor, aceder ao Treq recebido e comparar com a data que colocou no seu autenticador, garantido assim que a resposta corresponde ao seu pedido. A mensagem enviada pelo servidor vai ser também intercetada novamente pelo MACHandler que irá novamente comparar os resumos, desta vez acedendo ao resumo colocado no header pelo servidor.

```
MAC Handler: Teste de integridade validado com sucesso. MAC iguais
Client Handler: Resposta do servidor valida, Treq validado
Message source: http://localhost:8080/binas-ws/endpoint
```

Abaixo apresentamos um exemplo das mensagens SOAP enviadas e recebidas pelo cliente ao executar um pedido de *rentBina*, representadas através do handler PrettyLogHandler

```
 OUT BOUND SOAP MESSAGE:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <t:ticket xmlns:t="http://ticket">CjxOaWNrZXQgeG1sbnM6bnMyPSJodHRwOi8va2VyYnkuc2Rpcy50ZWNuaWNvLnVsaXNib2EucHQvIj4KICAgIDxkYXR
8zd1MySTVkc0RmdDVJY2VGT1QxQkFNQWRZbkZwQXFaamllTzJIZTRtdVlMeHdmaDhJUzVoL044NEE0ZE9xd215c3VBZnFXSWdLaFBWNmg2QjNKSXZ2QjRnN2JQelQzeVU
hBN09jOW93K2U1Q09EZ1ZLNEtITlVSNi93ZUZKMnFrNmF0bU1PTk12cHQyeDByUG1SYjFZYnBMRmRSaVcvQTdSRTN4M0xkZE5EwUxUVDR6NWo5TVBMbGVmampuUENUU1d
4KPC90aWNrZXQ+Cg==</t:ticket>
    <a:auth xmlns:a="http://auth">CjxhdXRoIHhtbG5zOm5zMj0iaHR0cDovL2tlcmJ5LnNkaXMudGVjbmljby51bGlzYm9hLnB0LyI+CiAgICA8ZGF0YT5zelU
t3RmhUNlgzL2x5N3ptY3FPelFheVhzaTdnWlJORDJmejJDYXVMbDhJS3FqdzhaQWtwcXRkWXl5SkcwZS96NDNKVi9YeHlSU21jeFJDMytoVmsyNzZYZ3NYbkdhNmRwTkN
RoPgo=</a:auth>
    <m:mac xmlns:m="http://mac">q8csHSstcLgwkyLv+Ophsgq8SuOwh56SQwmaKyvNeMI=</m:mac>
  </SOAP-ENV:Header>
  <S:Body>
    <ns2:rentBina xmlns:ns2="http://ws.binas.org/">
      <stationId>A67_Station1</stationId>
      <email>alice@A67.binas.org</email>
    </ns2:rentBina>
  </S:Body>
</S:Envelope>
```

```
 IN BOUND SOAP MESSAGE:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <m:mac xmlns:m="http://mac">82sR+cd8YxWBLKqabKlKmVr+6vQf7gFPDO8DwE8qlqc=</m:mac>
    <r:treq xmlns:r="http://treq">CjxOcmVxIHhtbG5zOm5zMj0iaHR0cDovL2tlcmJ5LnNkaXMudGVjbmljby51bGlzYm9hLnB0LyI+CiAgICA8ZGF0YT5jdOc2M1pH
R3MVZaWWtNRXJlbkNKL3VyMO42bkpuVFVMNGUwwExrcjBJSOh6cEduOXVJelVoc1pZMk5wQ3d4MEdvSHRDWStPWWYONTk4c21zcFBjdFdNVk5UVitGSmx3Y3FGSW1ETWJ1YkxiM
  </SOAP-ENV:Header>
  <S:Body>
    <ns2:rentBinaResponse xmlns:ns2="http://ws.binas.org/"/>
  </S:Body>
</S:Envelope>
```

Caso F2: Demonstração de resistência a um ataque

Cenário: Atacante tenta enviar uma mensagem antiga para o servidor, tentando um ataque de replay

Para ilustrar como o nosso programa está preparado para este tipo de ataque, vamos ilustrar a seguinte situação: Alice tenta enganar o servidor ao enviar uma mensagem antiga de modo a tentar executar a mesma operação. Para simular esta situação, iremos colocar no autenticador criado pelo cliente uma data de Janeiro em vez da data atual:

```
// Cria o autenticador
//Date currDate = new Date();
Calendar cal = Calendar.getInstance();
cal.set( year: 2018, Calendar.JANUARY, date: 9, hourOfDay: 10, minute: 11, second: 12); //Year, month, day of month, hours, minutes and seconds
Date currDate = cal.getTime();

Auth auth = new Auth(VALID_CLIENT_NAME, currDate);
smc.put("authDate", currDate);
smc.setScope( name: "authDate", MessageContext.Scope.APPLICATION);
CipheredView cipheredAuth = new CipheredView();
try {
    cipheredAuth = auth.cipher(sessionKey);
} catch (KerbyException e) {
    e.printStackTrace();
}
```

São então executados os passos descritos no caso anterior, e quando chega ao **KerberosServerHandler,** este irá obter o autenticador no header da mensagem, decifrá-lo com a sua chave de sessão e verificar se o timestamp está dentro da validade aceitável, entre T1 e T2. Como neste caso colocámos um data bastante anterior, o servidor vai detectar que a mensagem não é fresca e vai interromper a execução com uma runtime exception. Tal resultado pode ser visto nas mensagens de output:

```
    </ns2:activateUser>
  </S:Body>
</S:Envelope>
Server handler Erro: O timestamp to auth não está entre T1 e T2Server handler: Caught exception in handleMessage:
```

Garante-se assim a proteção contra ataques de replay.