FCT-NOVA

Segurança de Redes e Sistemas de Computadores

Relatório

A Secure REST-Based Messaging Repository System with Mutual TLS Client/Server Authentication and Access Control

Autores
Jorge Pereira nº 49771
Rodrigo Lopes nº 50435
Professor:
Henrique João Lopes Domingos
Turno: P1

11 de Dezembro de 2019

Resumo

Este projecto tem como objectivo implementar um repositório de mensagens seguro, não só do ponto de vista de transporte de mensagens, onde estas, iram protegidas com cipher-suites descritas ao longo deste relatório, mas também contra ataques internos de possíveis funcionários honestos mas curiosos (Honest-but-curious), que com as nossas medidas e implementações de segurança, fará com que este tipo de ataques não sejam possíveis. Decidimos usar a framework Spring tanto para a implementação do servidor, como para a do cliente.

Conteúdo

1 Introdução

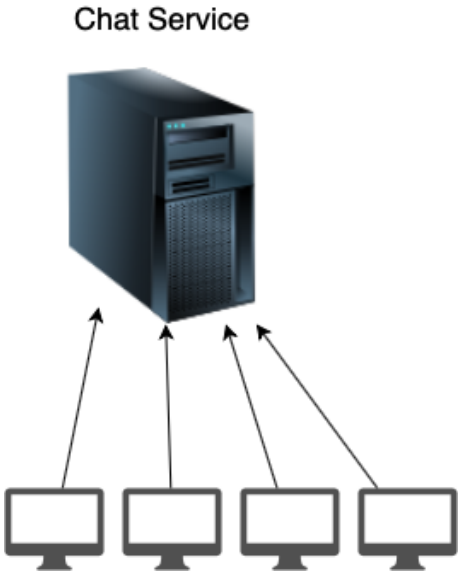
Este projecto terá como principais, um servidor, que será um repositório de mensagens cifradas pelos clientes de forma a ser impossível, a quem tenha acesso à base de dados, ver o conteúdo das mensagens. O servidor foi desenvolvido em Spring e com algumas modificações daquilo que seria a proposta inicial da API. Toda a parte criptográfica ficou no lado do cliente, sendo este o principal que cifra a mensagem e está encarregado de a decifrar e assinar de acordo com as suas configurações de segurança. Para o handshake inicial entre o servidor e o cliente, usamos TLS que utiliza as mesmas configurações (certificado e chaves) que são usadas para a cifragem das mensagens a enviar ao servidor, de modo a ser mais fácil o desenvolvimento do projecto. Por fim iremos apresentar a nossa pequena auditoria de segurança ao nosso servidor de modo a perceber e ter uma ideia prática das ferramentas e metodologias que são usadas nos dias de hoje para esse fim.

2 Modelo do Sistema e Arquitectura

2.1 Modelo do Sistema

Numa análise muito superficial do sistema, podemos visualizar o sistema como um repositório que contém uma dada interface para executar invocações remotas, como podemos visualizar na figura 2.1. Estas invocações permitem aos utilizadores enviarem mensagens para outros utilizadores, receberem mensagens, enviarem recibos de leitura, verificar o estado de uma mensagem, entre outras. O servidor serve apenas de repositório onde os utilizadores executam acções, semelhante a um sistema de publish/subscriber mas, que cada utilizador tem o seu tópico.

Figura 2.1: Vista superficial do sistema

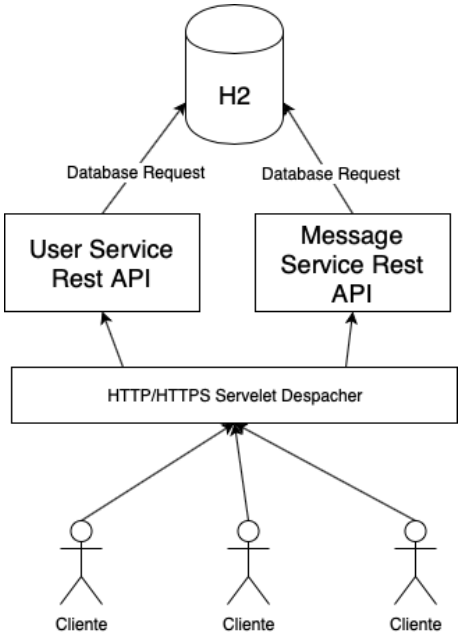


Esta interface para as operações remotas está implementada sobre *REST*. Contendo dois serviços (um para operações relacionadas com os utilizadores e outro para operações relacionadas com operações de mensagens). Como estamos a usar a *Spring framework* existe um Http Servelet Dispatcher que

seleccionará, dependendo do tipo do caminho do recurso, a qual serviço entregar para este posteriormente, responder. Para guardar de forma persistente as mensagens dos utilizadores e até mesmo os utilizadores, é usada uma base de dados SQL (H2).

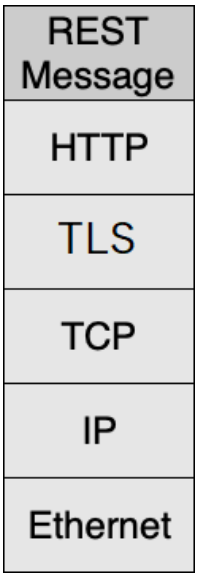
Vejamos a figura 2.2:

Figura 2.2: Vista interna do sistema



As comunicações entre o cliente e o servidor de forma a garantir confidencialidade, integridade e autenticidade dos dados passados são executadas sobre TLS. Vejamos o stack de protocolos usados presente na figura 2.3.

Figura 2.3: Stack de Protocolos usados



2.2 Arquiterura

Na nossa arquitectura foi utilizada a Spring Framework, onde implementamos a nossa API para o nosso Server Repository. Na API dividimos os nossos serviços em dois, o das mensagens e o dos utilizadores que serão especificados mais detalhe no ponto 3.1. Para a base de dados, decidimos utilizar H2 de forma a facilitar a manipulação e visualização dos dados numa maneira rápida e intuitiva.

Também para o cliente, utilizámos a Spring Shell, que nos forneceu facilidade no desenvolvimento do nosso cliente, tendo sido uma excelente aprendizagem neste tipo de tecnologias.

2.3 Modelo de Adversário

O servidor e o cliente em cooperação deverão respeitar o seguinte modelo de adversário:

- Confidencialidade de Mensagens; Integridade e autenticação das mensagens, Escuda das mensagens passadas na rede; Modificação e introdução de mensagens;

Também deverá ser garantida a não repudiação de um recibo de leitura tal como a autenticidade da mesma e a confidencialidade das mensagens durante todo o seu tempo de vida.

Para conseguir suportar os requisitos desde modelo foram usadas diversas estratégias para as diferentes fases do ciclo de vida de uma mensagem.

Durante a transmissão:

Durante a transmissão será necessário garantir a confidencialidade, integridade, a escuda e a modificação ou introdução de mensagens. Para isso foi usado o protocolo de TLS que irá criar um tunel seguro entre o cliente e o servidor. Um atacante acima do protocolo *TLS* não irá conseguir visualizar que tipo de tráfego está a passar na rede. Qualquer tentativa de introdução ou alteração de mensagens será detectado pelo *TLS* e alertado, caso seja grave, para o termino da comunicação *TLS* corrente.

Guardada no repositório e posterior envio para cliente:

De forma a garantir a confidencialidade das mensagens quando estão guardadas no repositório do servidor, é necessário que estas estejam a usar um envelope para esconder o seu conteúdo e garantir a sua integridade contra administradores de sistema que possam ter acesso a estas mensagens (agentes curiosos). Este envelope também ajudará a defender contra ataques que um cliente tente passar por outro e obter as mensagens deste.

A estrutura do envelope é a seguinte para o receptor:
$\{ks\}_{kpubReceptor}||\{M\}_{ks}||DigitalSignature_{KpubSender}$

Quem envia a mensagem também precisará de ter uma estrutura semelhante de modo a que possa mais tarde saber o conteúdo enviado. Vejamos o envelope:

$\{ks2\}_{kpubSender}||\{M\}_{ks_2}||DigitalSignature_{KpubSender}$

Envio prova de leitura da menagem:

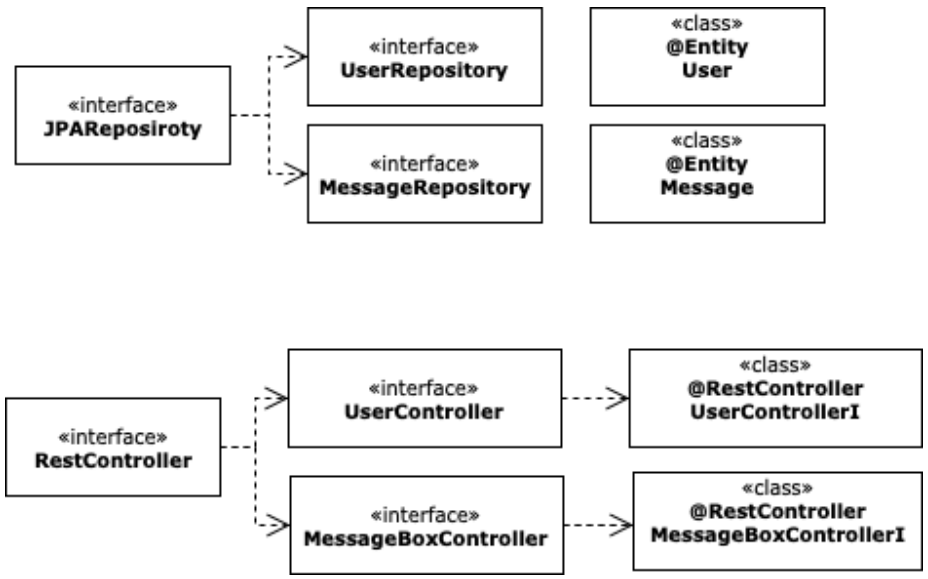
$DigitalSignature_{kpubReceptor}(ReceptorEvelope)$

3 Detalhes da implementação

Como foi dito anteriormente, foi usado *Spring* com o modulo *Spring Boot* para a implementação do servidor e do cliente, assim para construir software que respeite as construções usadas nessa *framework*, foi necessário usar o padrão de desenho MVC (Model View Controller). Assim foi necessário dividir as componentes do trabalho em Model's (User,Message) , Controller's (UserController , MessageBoxController) o View não está presente mas se fizermos analogia seria a linha de comandos do cliente. Também foi necessário oferecer componentes de persistência de dados para comunicarem com a base de dados escolhida. Estes componentes são os Repositórios (UserRepository, MessageRepository).

Utilizamos uma *in-memory database* para ser só necessário instanciar um serviço no sistema de Cloud da Microsoft (o Azure).

Figura 3.1: Componentes



No modo geral todos os modelos são convertidos JSON e colocados no payload de uma mensagens HTTP. Vejamos os campos de cada modelo de dados.

Uma mensagem:

- **id** - O ID representa um número único na base de dados de cada mensagem
- **from** - ID do utilizador que envia a mensagem
- **to** - ID do destino da mensagem, ou seja do receptor
- **messageFrom** - Conteúdo da mensagem para o remetente (Envelope cifrado)
- **messageFrom** - Conteúdo da mensagem para o receptor (Envelope cifrado)
- **sendDate** - Data de envio da mensagem (data de recepção poe parte do servidor)
- **receivedDate** - Data de recepção da mensagem (quando o receptor envia recibo de leitura)
- **signature** - Recibo de leitura por parte do receptor. (Assinatura digital do messageTo com a sua chave privada)
- **parameters** - Parâmetros de segurança usados (SAP - Security Association Parameters)

Exemplo de uma representação em JSON da mensagem:

```
{
  "id": "123",
  "from": "10",
  "to": "15",
  "messageFrom": "0cQ3CrNyiuuKBdlLL1mV+gAAABNgOMNKEYmnKEyyYnySF0K
     0OIXr|RkehQZTbay0JW...",
  "messageTo": "o9IQKnLdwyyEOyzs0WQ4xgAAABMcgC6xKivMm/bkiXM+/4
     NTXmXK|AHJhGMHHBV+GZUF...",
  "receivedDate": null,
  "signature": null,
  "parameters": ["ASSYM":"AES", "ASSYMKEY":"128", ...]
}
```

Um Utilizador:

- **id** - Id único gerado pelo servidor
- **uuid** - Identificador único de utilizador (Hash da chave publica)
- **secdata** - Chave publica do utilizador

```
{
  "id": "1",
  "uuid": "udacK9TDHr5jVFne6cfjVGlBx0WqrF5FFSKAxVWjOXw\u003d",
  "secdata": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn+DzG5
     ATsJRHwiiFFdMzngtzns..."
}
```

3.1 Especificação da API

Os endpoints do nosso servidor estão divididos em dois tipos de serviços:

- **API para as mensagens**
- **API para os utilizadores**

3.1.1 API das mensagens

Aqui fica toda a logística e tratamento das mensagens guardadas no nosso servidor. O tipo de funções que este serviço tem, são os listados no enunciado, como:

- Listar as mensagens ainda não lidas pelo utilizador (NEWS).
- Listar todas as mensagens de um utilizador (ALL).
- Enviar uma nova mensagem (SEND).
- Receber uma mensagem (RECV).
- Assinar uma mensagem de forma a registar a sua leitura (RECEIPT).
- Ver o estado de uma mensagem (STATUS).

3.1.2 API dos utilizadores

Neste serviço estão implementadas as funcionalidades dedicadas aos utilizadores, tais como:

- Criar um novo utilizador no sistema (CREATE).
- Listar todos os utilizadores inseridos no sistema (LIST).

3.2 Configurações de TLS

Para configuração do TLS poderá alterar-se as propiedades do ficheiro *application.properties*:

server.port= 8443 - indica a porta a que o servidor está ligado.

server.ssl.enabled= true - activar ssl/tls

server.ssl.key-store= test.jks - keychain com chave privada e chain de certificados do cliente

server.ssl.key-password= password - password da chave privada

server.ssl.key-store-password= password - password da keystore

server.ssl.client-auth= need - caso seja necessário autentificação do cliente.

server.ssl.enabled-Protocols: [TLSv1.2] - protocolos aceites de tls pelo servidor

server.ssl.ciphers:"TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256"]

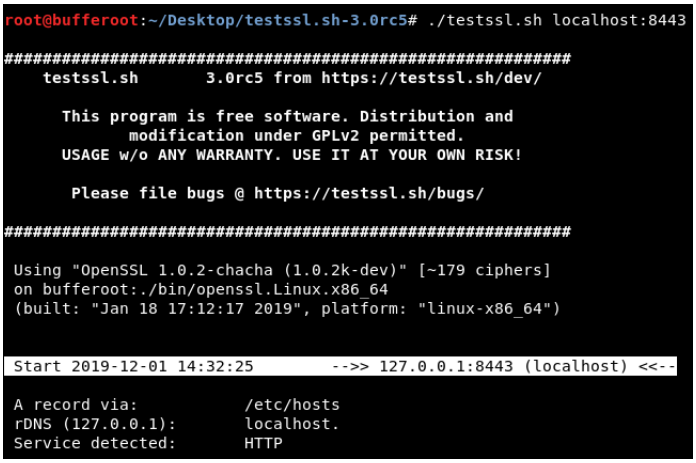
4 Validação e Avaliação da solução

4.1 Auditoria de Segurança ao servidor

Para tornar o nosso relatório um pouco mais completo, decidimos adicionar este capitulo, que basicamente serve para testar-mos algumas ferramentas que testam e encontram possíveis vulnerabilidades ao nosso servidor, e assim conseguirmos, mais rapidamente, encontrar alguns erros para mais tarde podermos corrigi-los. Por falta de tempo não nos foi possível melhorar alguns erros dados como críticos, mas sem dúvida que entende-los e perceber-los foi uma importante aprendizagem.

Iremos começar por mostrar o inicio da nossa auditoria, onde corremos uma ferramenta que verifica os serviços de um dado servidor e os seus suportes de cifras TLS/SSL, protocolos e possíveis falhas e vulnerabilidades criptográficas.

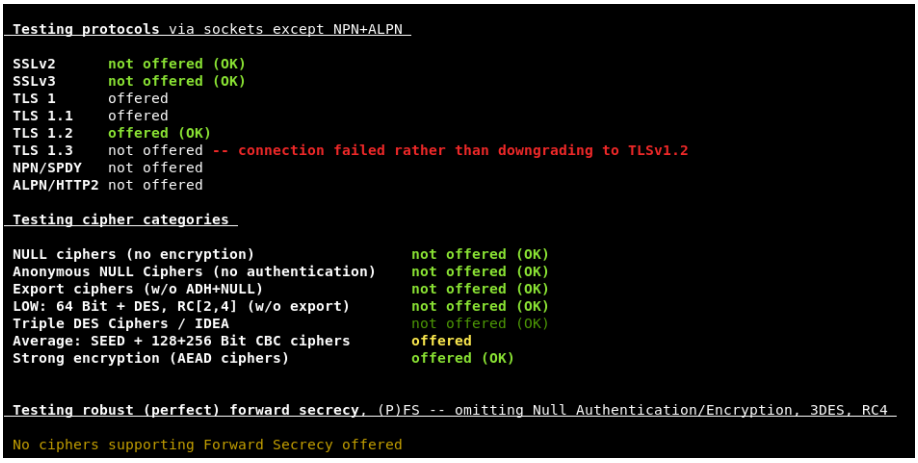
Figura 4.1: testssl.sh localhost:8443



Como podemos ver na figura seguinte, o nosso servidor oferece TLS 1 a TLS 1.2, não oferecendo SSLv2 ou SSLv3 do que no ponto de viste deste teste é bom.

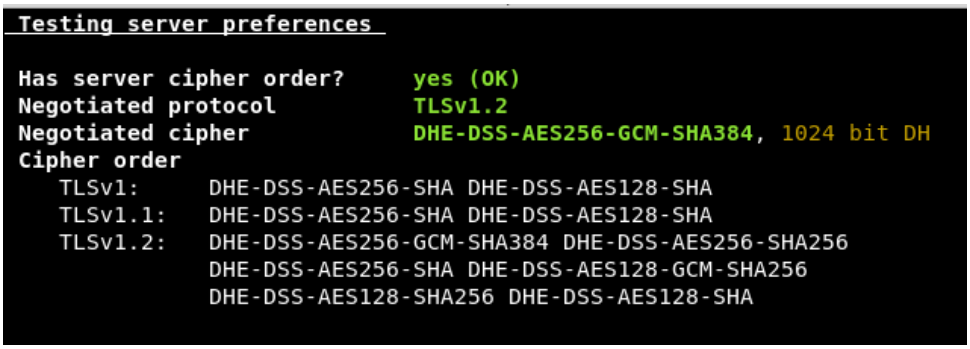
Testing Robust (perfect) forward secrecy /TODO/

Figura 4.2: Teste de protocolos e cifras.



Aqui podemos ver que para todos os protocolos aceites pelo servidor, têm como cipher suits "preferidas"Diffie Hellman, do que do ponto de vista de segurança está correto e seguro para este teste.

Figura 4.3: preferencias do servidor (cipher suits).

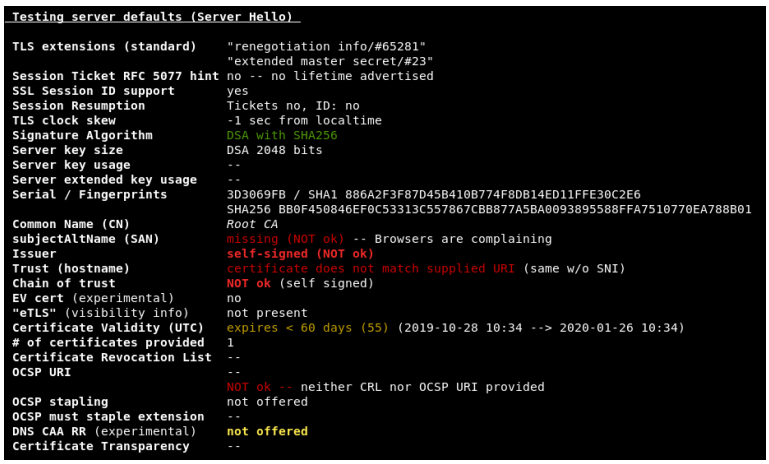


Com este teste reparamos que alguns parâmetros nos certificados que geramos, estavam em falta e outros "inseguros", o que rapidamente entendemos o porquê, pois dada a maneira como geramos os certificados era perfeitamente normal este tipos de "vermelhos". Alguns destes erros podem ser resolvidos utilizando a ferramenta openssl com o seguinte comando fornecido pelo professor:

```
$ openssl req
    -new
    -sha256
    -key domain.key
    -subj "/C=US/ST=CA/O=Acme, Inc./CN=example.com"
    -reqexts SAN -config <(cat /etc/ssl/openssl.cnf
```

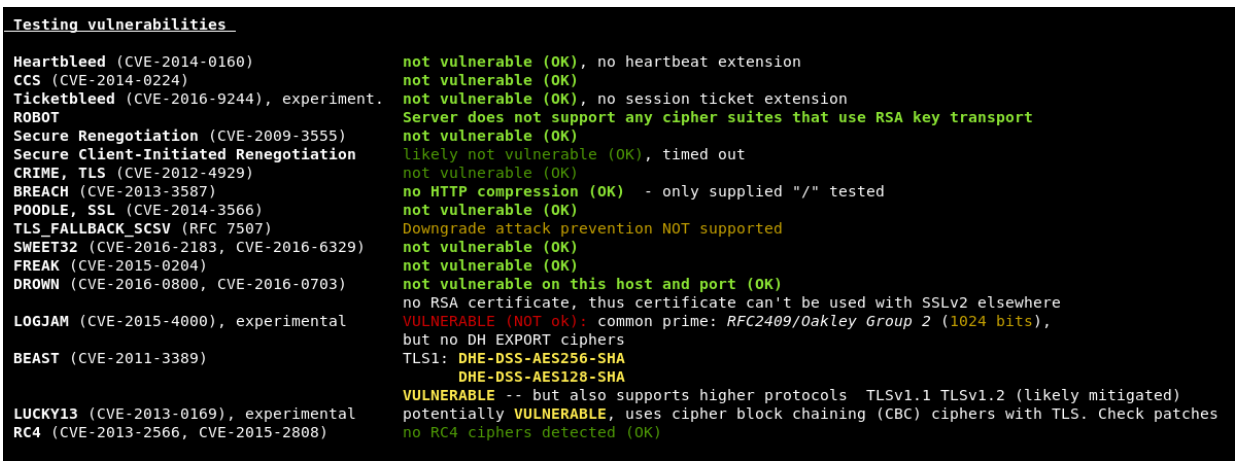
```
<(printf "\n[SAN]\nsubjectAltName=DNS:example.com,DNS:www.example.com")
-out domain.csr
```

Figura 4.4: Teste Server Hello.



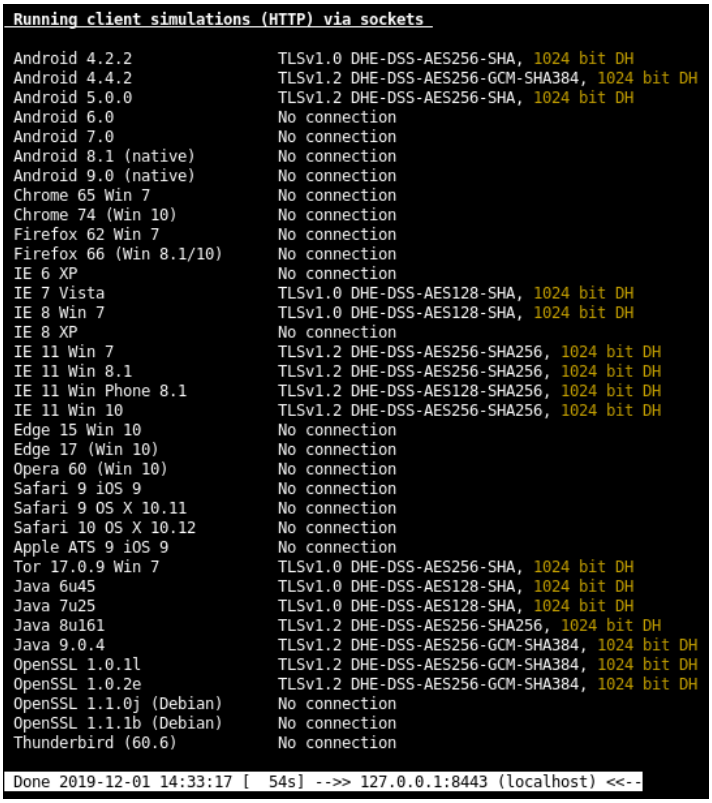
No teste de vulnerabilidades, podemos notar que praticamente todos os testes deram "luz verde", excepto um, LOGJAM, que é uma vulnerabilidade contra a troca de chaves usando Diffie Hellman. Para resolver este problema, teríamos que perceber melhor como este tipo de ataque funciona e tentar encontrar uma solução.

Figura 4.5: Testes de vulnerabilidades.



Na imagem seguinte estão apenas alguns testes fornecidos por este script, para ver-mos quais as cipher suits acordadas por cada tipo de plataforma. Sendo possível ver que todas as plataformas que conseguiram estabelecer conexão com o nosso servidor, acordaram em usar Diffie Hellman.

Figura 4.6: Simulação de vários tipos de clientes.



Use this section to discuss the validation and correctness of your designed system and related implementation (prototype). Explain the experimental evaluation done, the considered, focused and observed evaluation criteria. Discuss how you evaluated or measured your system for those criteria (referring experiments, practical observations/deployment and possible qualitative, as well as, quantitative metrics you’re your performed observations). If you want to structure more clearly the section, can use an initial paragraph describing the evaluation and validation objectives as addressed, dedicating a sub-section to specific observations done and argumentation about the system validity from your observations.

5 Conclusão

Em suma, este projecto revelou ser quase um full stack project, onde desenvolvemos um serviço seguro que nos obrigou a aprender um pouco mais não só sobre o tema da cadeira (segurança) mas também sobre desenvolvimento de uma API, usando a Spring Framework, manipulação de base de dados SQL, usando Spring com H2, e por fim, ainda descobrimos dentro da Spring Framework, a existência da Spring Shell que nos facilitou imenso na construção do nosso cliente.

Em termos dos objectivos da cadeira, foram todos os requisitos propostos, concluídos, dentro do tempo estimado. Por motivos de falta de tempo, não nos foi possível desenvolver um front-end mais apelativo para o cliente, mas a nosso ver o nosso shell client vai de acordo com o pretendido.

A

Tabela de Sumário da implementação do trabalho TP2 submetido para avaliação