Ciências ULisboa | Informática

# Aplicações Distribuídas

## 2015/16

## Projeto 4

### 1. Descrição geral

A avaliação da cadeira de aplicações distribuídas está divida em quatro projetos. O quarto projeto incide sobre medidas de desempenho e de segurança para o projeto anterior.

No quarto projeto foi concretizado um serviço WEB para gerir um sistema simplificado de inscrição de alunos em turmas. Para esse efeito no servidor foi utilizada a *framework* de desenvolvimento WEB *Flask* [2] e o motor de base de dados SQL *sqlite* [3]. O programa cliente utiliza o módulo *requests* [4] para implementar a interação cliente/servidor baseada no HTTP.

### 2. Desempenho

Assumindo que o serviço é executado através do servidor WSGI embutido no *Flask* e que este executa num computador com múltiplos núcleos de processamento, os alunos deverão alterar o servidor do projeto 3 para que este responda eficientemente a múltiplos clientes. Além das alterações ao código deverão justificar o método escolhido no ficheiro README que será entregue.

Ainda no sentido de aumentar o desempenho, independentemente das capacidades ou configuração do servidor, o cliente desenvolvido através do módulo *requests* deverá fazer com que os vários pedidos feitos ao servidor sejam enviados na mesma ligação TCP (ligações persistentes no HTTP).

### 3. Autenticação e confidencialidade da comunicação

O protocolo SSL/TLS deverá ser utilizado com o *Flask* e com o módulo *requests* para que o cliente e o servidor verifiquem mutuamente a autenticidade do interlocutor e para que a comunicação seja confidencial (cifrada). Para este efeito, cliente e servidor deverão ter certificados de chave pública assinados por uma *Certificate Authority (CA)*. A implementação no *Flask* será feita através da classe `SSLContext` do módulo `ssl` [1] da biblioteca padrão do *Python*.

#### 3.1. Criação de certificados

Para que os certificados do cliente e do servidor possam ser assinados por uma CA é necessário criar um certificado para a CA fictícia:

**`openssl genrsa -out root.key 2048`**
(cria a chave da CA)

**`openssl req -x509 -new -nodes -key root.key -sha256 -days 365 -out root.pem`**
(cria um certificado autoassinado para a CA)

De seguida serão geradas as chaves do servidor e do cliente de acordo com o seguinte padrão de comando:

**`openssl genrsa -out <nome do ficheiro>.key 2048`**

Para que a CA assine os certificados do cliente e do servidor ter-se-á que emitir um pedido de assinatura de certificado para cada um. Isso pode ser feito de acordo com o padrão de comando:

```
openssl req -new -nodes -key <nome do ficheiro>.key -sha256 -days 365 \
-out <nome do ficheiro>.csr
```

Na posse dos pedidos de assinatura a CA procede à geração dos certificados assinados por si. Isso pode ser feito para o cliente e para o servidor pelo seguinte padrão de comando:

```
openssl x509 -req -in <nome do ficheiro>.csr -CA root.pem -CAkey root.key \
-CAcreateserial -out <nome do ficheiro>.crt -days 365 -sha256
```

Através deste método cliente e servidor deverão, cada um, ter um par de ficheiros `<nome>.crt` e `<nome>.key`. Estes serão utilizados nos programas para que a autenticação e comunicação cifrada com o interlocutor sejam possíveis. Na implementação da autenticação os programas cliente e servidor terão que utilizar o certificado da CA (root.pem) para validar a assinatura do certificado apresentado pelo interlocutor.

## 4. Configurações de Segurança

Além das tarefas descritas acima, os alunos deverão definir regras do *iptables* (*firewall*) e do *snort* (deteção e prevenção de intrusões) para proteger o servidor do sistema.

### 4.1. Firewall *iptables*

**Antes de começar a realizar o projeto, estude a ferramenta *iptables* e efetue os exercícios do guião da aula PL.**

Pretende-se que os alunos utilizem o comando *iptables* de modo a configurar a máquina segura onde será instalado o servidor.

A melhor maneira de garantir a segurança da máquina é reduzir os seus serviços ao mínimo indispensável e garantir a sua constante atualização. Neste contexto, a *firewall* deve ser configurada de modo a concretizar a seguinte política:

- Serviços suportados: *ping*, serviços necessários para o servidor e serviço SSH.

  Restrições: a máquina responde a *pings* com origem na máquina **nemo.alunos.di.fc.ul.pt**, aceita ligações de clientes com qualquer origem para o servidor, e aceita ligações SSH da sua rede local.

- Serviços utilizados: DNS

Os alunos devem incluir no ficheiro README um relatório com o seguinte conteúdo:

- Regras do comando *iptables* que permitem concretizar a política; e
- explicação do método de teste utilizado e observações realizadas.

**Observações**:

i) o normal funcionamento dos computadores dos laboratórios depende do seu acesso às seguintes máquinas:
   DCs: 10.101.253.11, 10.101.253.12, 10.101.253.13
   Storage: 10.101.249.63
   Iate/Falua: 10.101.85.6, 10.101.85.138
   Nemo: 10.101.85.18
   Gateway: 10.101.148.1
   Proxy: 10.101.85.134
   Deste modo ao testarem as suas regras não devem impedir o acesso a estas máquinas.

ii) a opção –F do *iptables* não altera a política definida por omissão. Assim, a seguinte sequência de comandos bloqueará o computador (ver justificação na observação anterior):

```
$…
$ sudo iptables –P OUTPUT DROP
$ sudo iptables –F OUTPUT
```

iii) O tráfego do dispositivo de loopback não deve ser filtrado:

```
$ sudo iptables -A INPUT -i lo -j ACCEPT
$ sudo iptables -A OUTPUT -o lo -j ACCEPT
```

iv) O tráfego relacionado com uma ligação já estabelecida também deve ser aceite:

```
$ sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED  -j
ACCEPT
$ sudo iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
```

### 4.2. Deteção de intrusões com *snort*

**Antes de começar a realizar este trabalho, estude a ferramenta *snort* e efetue os exercícios da aula PL.**

Pretende-se que os alunos utilizem o *snort* de modo a detetarem alguns ataques contra a máquina onde o servidor está em execução. Os alunos devem definir uma ou mais regras *snort* para as situações seguintes, potencialmente indicativas de um ataque:

- Deve ser gerado um alerta para a consola quando forem recebidos no servidor 10 ou mais ligações TCP provenientes da mesma máquina emissora, para portos inferiores a 1024 durante um intervalo de dois minutos (pode indicar um varrimento de portos). (NOTA: nesses dois minutos deve ser gerado **apenas** um alarme por máquina que faz o ataque).
- Deve ser gerado um alerta para a consola sempre que forem vistas 4 ligações da mesma máquina emissora para o porto do servidor, durante um intervalo de 30 segundos (poderia indicar que estão a tentar descobrir uma password de acesso ao serviço) (NOTA: deve haver um alerta **por cada** conjunto de 4 ligações observadas).

Os alunos devem incluir no ficheiro README um relatório com o seguinte conteúdo:

- regra(s) definida(s) para o comando *snort* com o comportamento descrito;
- forma de invocação do comando *snort*
- método de teste utilizado e observações realizadas

## 5. Entrega

A entrega do projeto 4 consiste em colocar todos os ficheiros *.py* do projeto numa diretoria cujo nome deve seguir exatamente o padrão **grupoXX** (por exemplo grupo01 ou grupo23). Juntamente com os ficheiros *.py* deverá ser enviado um ficheiro de texto README.txt (não é .pdf nem .rtf nem .doc nem .docx) onde os alunos devem incluir a informação pedida no enunciado e onde podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). Os certificados devem ser incluídos numa subdiretoria de **grupoXX**. Esta última será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão **grupoXX.zip**. Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL.

Note que a **entrega deve conter apenas os ficheiros .py, .sql, certificados e o ficheiro README.txt, qualquer outro ficheiro vai ser ignorado**.

**O prazo de entrega é sábado, dia 21/05/2016, até às 22:00hs.**

## 6. Bibliografia

[1] https://docs.python.org/2/library/ssl.html