Relatório do Sprint 1 de ASIST (Administração de Sistemas)

Elementos do grupo:

André Novo, 1181628

Diogo Ribeiro, 1180782

Hugo Frias, 1170500

João Cruz, 1161255

Vera Pinto, 1180730

Introdução:

Este relatório contém a forma como o nosso grupo interpretou os casos de uso requisitados e a forma como os mesmos foram realizados.

Realização das User Stories:

User Story 1:

Enunciado:

Como administrador da infraestrutura quero que o servidor Windows e Linux forneçam endereços IP (na segunda placa de rede) da família 192.168.X.0/24 aos postos clientes, onde X é obtido por 100 + número_do_grupo (exemplo, para o grupo 99, X=199).

Realização em Linux:

Nas aulas teórico-práticas, já tinha sido instalado o servidor DHCP, usando o comando:

```
sudo apt install isc-dhcp-server
```

O nosso grupo já tinha criado a segunda placa de rede, com o nome ens192. No entanto, mudámos o endereço para um dos endereços da rede 192.168.126.0/24. No caso, escolhemos o endereço 192.168.126.1. Editámos o ficheiro **/etc/netplan/50-cloud-init.yaml** com a seguinte configuração:

```
    ens192:
        addresses:
            - 192.168.126.1/24
        dhcp4: no
        dhcp6: no
```

Num primeiro momento, o grupo editou o ficheiro de configuração do servidor DHCP **/etc/dhcp/dhcpd.conf**, adicionando uma declaração subnet à interface ens192, que contém

um range correspondente ao intervalo de endereços da família 192.168.126.0/24. Cada posto cliente irá receber um dos endereços do intervalo por UDP. As declarações options dizem respeito aos parâmetros que serão enviados aos postos clientes via DHCP. O ficheiro ficou com a seguinte configuração:

```
option domain-name "dei.isep.ipp.pt";
option domain-name-servers 192.168.62.32, 192.168.62.8;

subnet 192.168.126.0 netmask 255.255.255.0 {
  range 192.168.126.3 192.168.126.254;
}

default-lease-time 600;
max-lease-time 7200;
```

O grupo usou o range completo da rede, excluíndo 2 endereços: 1 deles corresponde ao da segunda interface em Linux e o outro em Windows.

De seguida, o grupo editou o ficheiro **/etc/default/isc-dhcp-server**, por forma a garantir que a interface ens192 é a única que irá fornecer informações via DHCP. Ou seja, a única interface que o servidor DHCP irá usar:

```
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="ens192"
INTERFACESv6="ens192"
```

Por fim, fizemos uso do seguinte comando, para arrancar com o servidor DHCP:

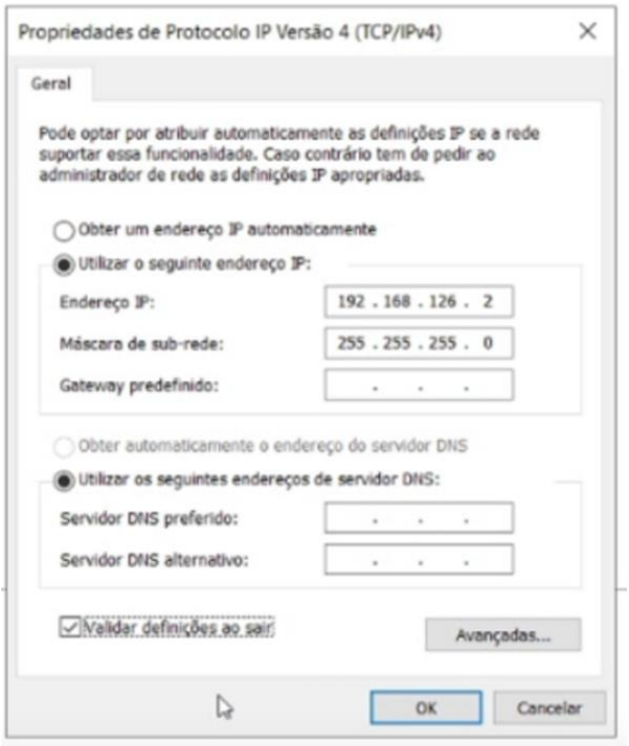
```
sudo servisse isc-dhcp-server start
```

Realização em Windows:

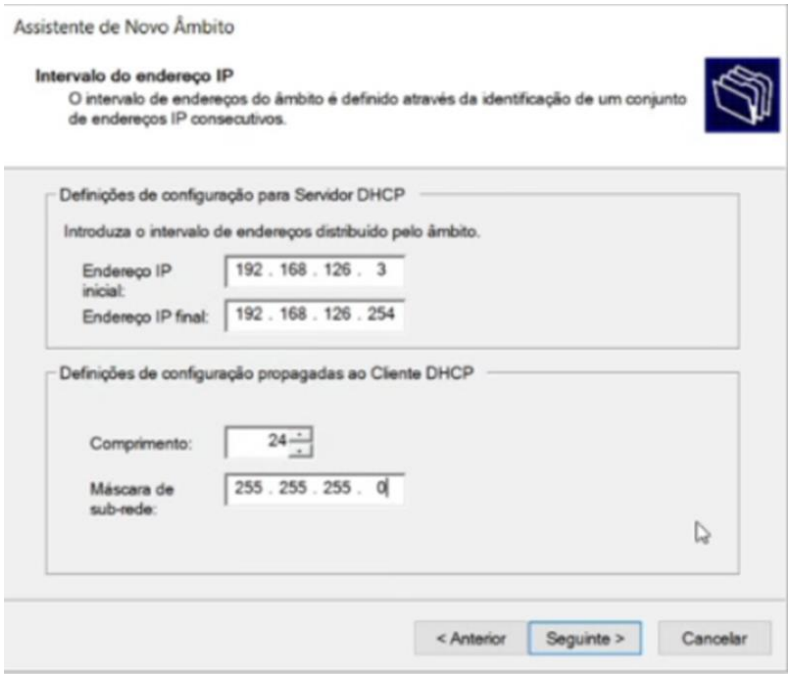
Na máquina Windows, o grupo definiu o endereço da segunda placa de rede Ethernet1 seguindo as seguintes instruções:

Painel de Controlo => Rede e Internet => Centro de Rede e Compartilhamento.

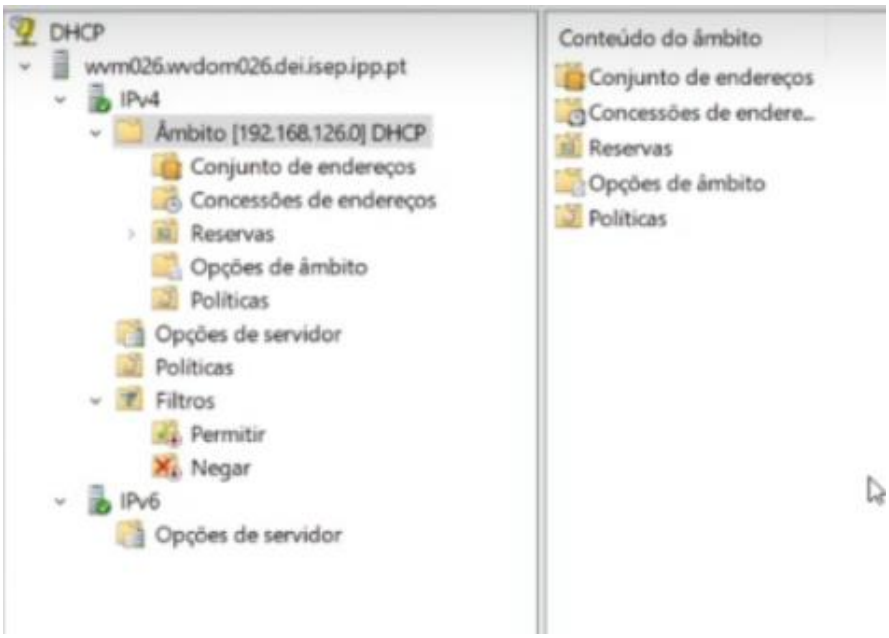
Nesta secção, selecionámos as propriedades da segunda placa de rede e, no campo referente ao protocolo de IP Versão 4 (TCP(IPv4), alterámos o endereço para o 192.168.126.2/24.



Em seguida, o grupo acedeu ao Gestor de Servidor e selecionou a secção DHCP. Ao nível do IPv4, criámos um novo âmbito, com o nome “DHCP” e com as seguintes caraterísticas:



O range de endereços definidos foi exatamente o mesmo definido para a máquina Linux.



User Story 2:

Enunciado:

Como administrador da infraestrutura quero que os serviços acima referidos funcionem em failover, com um deles a facultar endereços de 192.168.X.50 a 192.168.X.150 e o outro de 192.168.X.151 a 192.168.X.200.

Realização em Linux:

Para que o servidor DHCP funcionasse em failover, o nosso grupo editou o ficheiro **/etc/dhcp/dhcpd.conf**, adicionando a seguinte declaração de failover:

```
failover peer "failover-partner" {
     primary;
     address 192.168.126.1;
     port 647;
     peer address 192.168.126.3;
     peer port 647;
}
```

Devido ao facto de não existir um segundo servidor DHCP em Linux se seria usado no caso do nosso servidor ter alguma falha, decidimos atribuir a esse servidor inexistente o endereço 192.168.126.3. A porta usada para a troca de pacotes entre os 2 servidores, no caso de algum deles falhar, é feita por TCP pela porta 647.

Para além da adição supracitada, o nosso grupo adicionou, na declaração subnet, uma secção pool, contendo a chamada da declaração failover e o range que será usado, caso o outro servidor tenha alguma falha ao fornecer IPs via DHCP. O range escolhido foi o seguinte: 192.168.126.50 até 192.168.126.150:

```
pool {
        failover peer "failover-partner";
        range 192.168.126.50 192.168.126.150;
}
```

Realização em Windows:

O nosso grupo decidiu não realização a parte deste User Story relativa à implementação do mesmo em Windows.

User Story 3:

Enunciado:

Como administrador da infraestrutura quero os servidores Windows e Linux estejam disponíveis apenas para pedidos HTTP e HTTPS. Tal não deve impedir o acesso por SSH ou RDP aos administradores (o grupo).

Realização em Linux:

Para a realização do que é pedido no enunciado, o grupo criou um script **(/etc/my.firewall**). O script irá conter as configurações da firewall. Todas as regras criadas para este US incidem sobre a tabela filter. Numa fase inicial, o grupo adicionou 3 linhas:

```
#!/bin/bash

iptables -P INPUT DROP
iptables -F INPUT
```

A primeira linha define a política da cadeia pré-definida INPUT (-P INPUT), que é uma cadeia que incide sobre todos os pacotes que são recebidos da rede e que têm como destino o nosso servidor. A política é aplicada sobre o pacote caso o mesmo não tenha nenhuma correspondência com nenhuma das regras definidas para a cadeia. No caso, os pacotes que sejam confrontados com essa política são bloqueados (DROP).

A segunda linha elimina todas as regras impostas anteriormente sobre a cadeira INPUT (-F INPUT).

As 2 linhas mencionadas em cima serão usadas, uma vez que iremos trabalhar sobre a cadeia INPUT.

Em seguida, adicionámos 3 regras à cadeia INPUT (-A INPUT) ao script:

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

A primeira regra permite que seja possível o acesso SSH à máquina, que é feito por TCP na porta 22. No entanto, a regra permite que qualquer utilizador consiga aceder por SSH à máquina, sendo que apenas queremos que os administradores consigam. Por isso, editámos o ficheiro **/etc/ssh/shhd_config**, adicionando a seguinte linha:

```
AllowUsers asist root
```

A linha basicamente diz que os utilizadores autorizados a fazer login via SSH têm que pertencer ao grupo especificado, no caso, **asist** e **root**.

A segunda e terceira linhas permitem que seja possível a máquina receber pedidos HTTP e HTTPS, respetivamente, nas portas 80 e 443, respetivamente.

Para pôr em prática as configurações definidas no ficheiro, o grupo executou o script.

Para que o servidor DHCP continuasse a correr, adicionámos mais 4 linhas ao script, adicionando regras não só à cadeia de INPUT, mas também à de OUTPUT, cadeia essa pré-definida também:

```
iptables -A INPUT -p udp --dport 67 -j ACCEPT
iptables -A OUTPUT -p udp --dport 68 -j ACCEPT
iptables -A INPUT -p tcp --dport 647 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 647 -j ACCEPT
```

As 2 primeiras linhas permitem o tráfego de pacotes UDP entre o servidor e o cliente DHCP, sendo que a porta 67 é usada para tráfego de pacotes do cliente para o servidor e a porta 68 para o tráfego contrário a esse.

As 2 últimas linhas já permitem o tráfego de pacotes TCP entre o nosso servidor e o outro servidor DHCP Linux (de momento, inexistente), na porta 647.

Por fim, corremos o script para aplicar todas as configurações feitas.

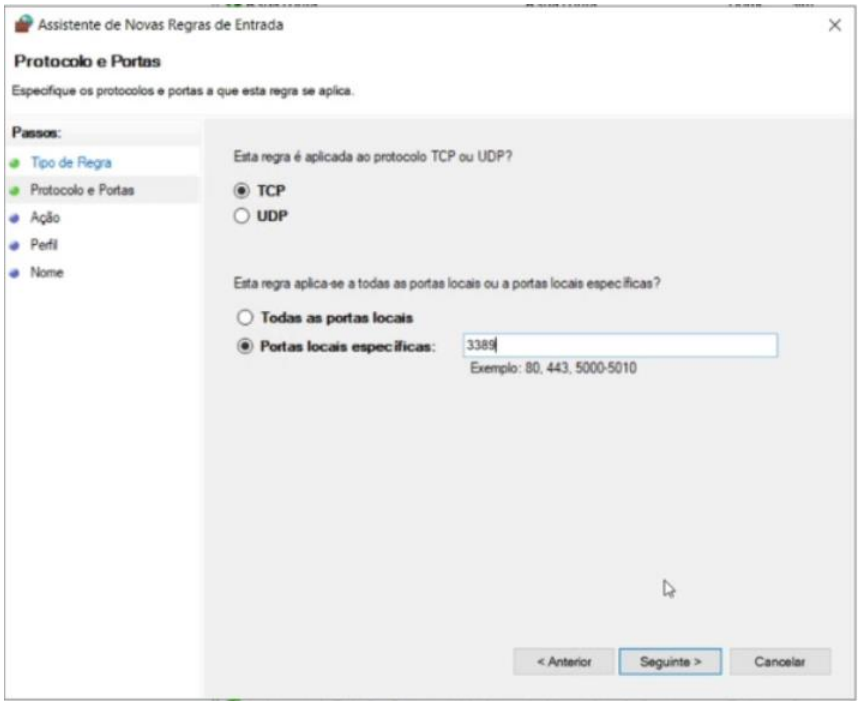
Realização em Windows:

Para a realização deste User Story em Windows, o grupo acedeu ao Windows Defenser Firewall, através do seguinte caminho:

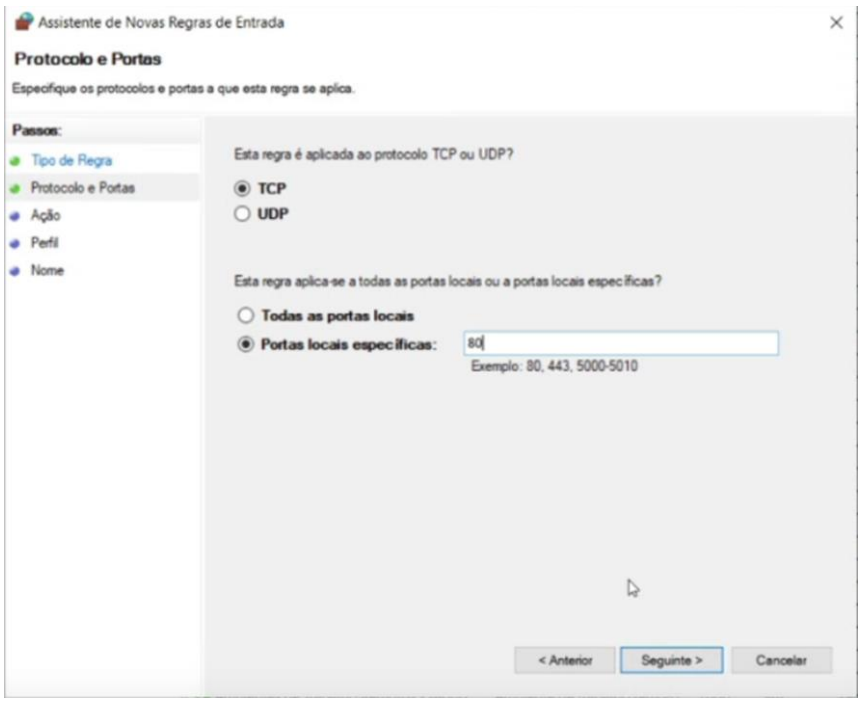
Painel de Controlo => Sistema e Segurança => Windows Defenser Firewall

Em seguida, adicionámos 3 regras de entrada:

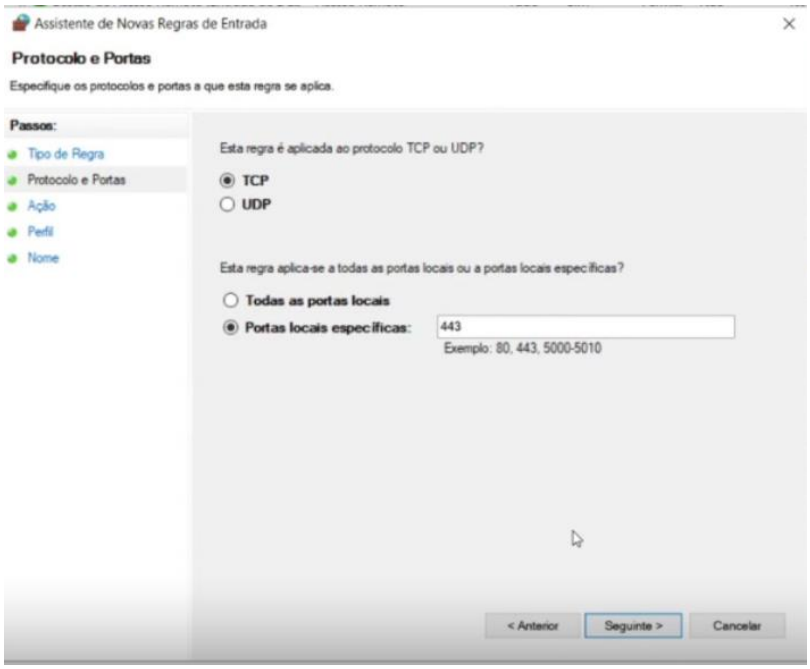
A primeira regra é referente à permissão de ligações RDP, na porta 3389 via TCP:



A segunda regra é referente à permissão de ligações HTTP, na porta 80 via TCP:



Já a terceira regra aplicada é referente à permissão de ligações HTTPS, na porta 443 via TCP:



User Story 4:

Enunciado:

Como administrador da infraestrutura quero impedir o IP spoofing na minha rede.

Realização em Linux:

O IP Spoofing pode ser de 2 tipos: interno e externo. Por forma a evitar cada um destes tipos, o grupo adicionou regras de firewall. Para evitar IP Spoofing interno, todos os pacotes que são mandados para a internet têm que possuir um endereço de origem que faça parte de uma das redes locais conhecidas. Já para evitar IP Spoofing externo, todos os pacotes provenientes da internet não podem possuir um endereço de origem que faça parte de uma das redes locais conhecidas.

Neste caso, a única rede local conhecida é a 192.168.126.0/24, que será a rede usada para comparar com o endereço de origem dos pacotes.

O grupo decidiu aplicar 1 regra de firewall para cada uma das interfaces (ens160 e ens192). Como a ens160 é a interface que recebe os pacotes provenientes da internet, o regra para evitar o IP Spoofing externo será aplicada sobre a mesma. Já a outra, como manda pacotes para a internet, terá uma regra para evitar o IP Spoofing interno.

Para o efeito, o grupo adicionou as tais 2 regras ao script de firewalls criado anteriormente, lembrando que todas as regras deste US incidem sobre a tabela filter:

```
iptables -A INPUT -i ens160 -s 192.168.126.0/24 -j DROP
iptables -A INPUT -i ens192 ! -s 192.168.126.0/24 -j DROP
```

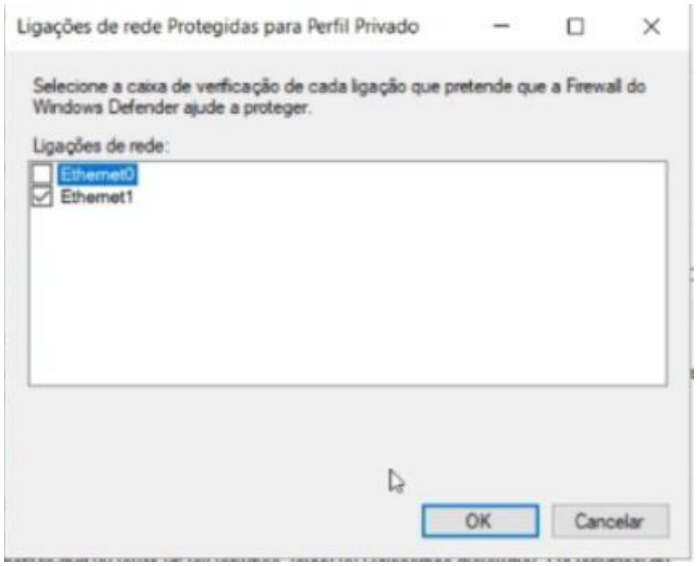
A primeira regra é aplicada a todos os pacotes que entram na interface ens160 e bloqueia-os caso o endereço de origem deles pertença à rede 192.168.126.0/24.

A segunda regra é aplicada a todos os pacotes que entram na interface ens192 e bloqueia-os caso o endereço de origem não pertença à rede 192.168.126.0/24 (para o efeito, foi usado o símbolo !).

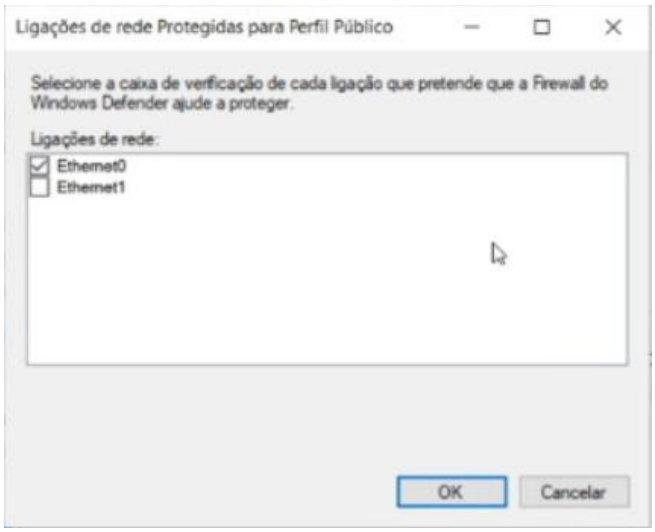
Realização em Windows:

Em Windows, o grupo acedeu, uma vez mais, ao Windows Defenser Firewall para aplicar as novas regras relativas ao impedimento do IP Spoofing na rede.

Num primeiro momento, o grupo definiu o endereço de rede que estaria associado tanto à rede pública como à rede privada. Para a rede privada, o grupo selecionou a segunda placa de rede, correspondente à interface Ethernet1 (192.168.126.0/24):

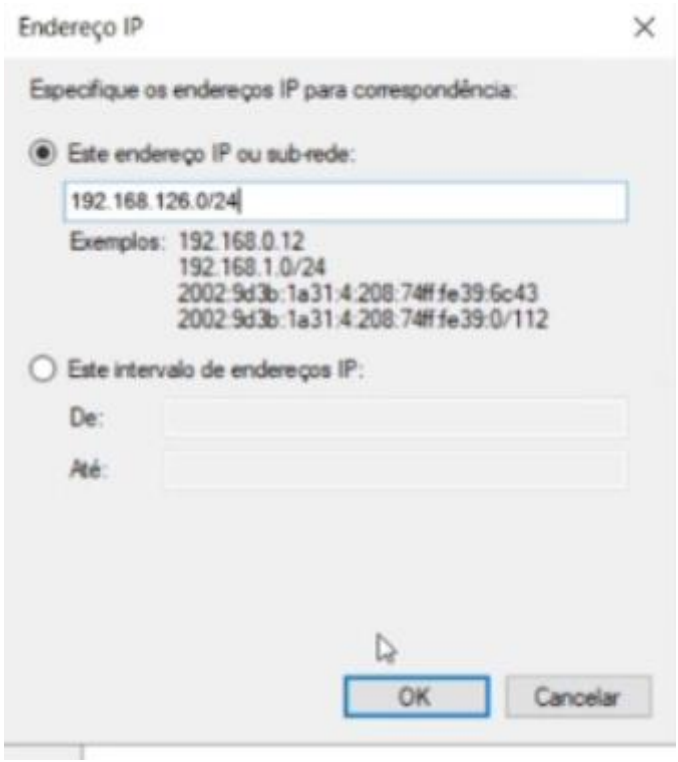


Já para a rede pública, o grupo selecionou a primeira placa de rede, correspondente à interface Ethernet0:



Em seguida, adicionámos 2 regras personalizadas: uma INBOUND e outra OUTBOUND. A primeira foi criada para evitar o IP Spoofing externo, enquanto que a segunda foi criada para evitar o IP Spoofing interno.

Ambas as regras fizeram uso da rede 192.168.126.0/24, para verificação dos endereços de origem.



User Story 5:

Enunciado:

Como administrador da infraestrutura quero que os utilizadores registados no Linux com UID entre 6000 e 6500 só consigam aceder via SSH se esse acesso for a partir de uma máquina listada em **/etc/remote-hosts**.

Realização:

Para a realização deste UC, o grupo criou um script, de nome “uc05”. O conteúdo do script é o seguinte:

```
#!/bin/bash

n1=1
n2=1
n3=1
n4=1
min=6000
max=6500
backupFile=/tmp/remote-hosts.prev
actualFile=/etc/remote-hosts
file=/etc/ssh/sshd_config
usersFile=/etc/passwd

while read line; do
userID=$(cut -d':' -f3 <<< "$line")
if [ "$userID" -gt "$min" ] && [ "$userID" -lt "$max" ]; then

        while read lineHost; do
        if ! grep -q -s "$userID@$lineHost" "$file"; then
                echo "AllowUsers $userID@$lineHost" >> "$file"
        fi
        n2=$((n2+1))
        done < "$actualFile"

        while read lineHostPrev; do
        if ! grep -q -s "$lineHostPrev" "$actualFile"; then
                sed -e "s/AllowUsers $userID@$lineHostPrev//g" -i "$file"
        fi
        n3=$((n3+1))
        done < "$backupFile"

fi
n1=$((n1+1))
done < "$usersFile"

while read actualFileLine; do
echo -n "" > "$backupFile"
echo "$actualFileLine" >> "$backupFile"
n4=$((n4+1))
done < "$actualFile"
```

Neste script, para cada UID presente no ficheiro **/etc/passwd** entre 6000 e 6500, percorremos o ficheiro **/etc/remote-hosts** e verificamos se o utilizador já tem permissão de aceder via SSH através de cada endereço do ficheiro, vendo se a linha **user@IP** já existe no ficheiro **/etc/ssh/sshd_config**. Se não existir, a permissão é adicionada. Caso contrário, nenhuma ação será realizada.

No entanto, um dos endereços definidos no ficheiro **/etc/remote-hosts** pode ser eliminado do ficheiro. Por culpa disso, criamos um ficheiro backup que guarda sempre uma versão desse ficheiro quando o script é corrido (**/tmp/remote-hosts.prev**).

Para verificarmos se algum elemento do ficheiro **/etc/remote-hosts** foi eliminado, comparamos ambos (comparamos cada endereço do ficheiro backup com os endereços atuais do ficheiro original). Se o endereço não existir mais no original, eliminamos o mesmo do ficheiro **/etc/ssh/sshd_config**.

De forma a evitar ter que correr manualmente o script, editámos o ficheiro **/etc/crontab** para correr o script de minuto a minuto:

```
*  *    * * *   root    /etc/uc05
```

User Story 6:

Enunciado:

Como administrador da infraestrutura quero que o acesso ao sistema seja inibido aos utilizadores listados em **/etc/bad-guys**.

Realização em Linux:

Para a realização deste UC, o grupo fez uso do módulo PAM, que consiste num conjunto de bibliotecas partilhadas usadas para autenticar dinamicamente um utilizador a aplicações ou serviços no sistema Linux.

Por forma a respeitar os requisitos do enunciado, o grupo editou o ficheiro **/etc/pam.d/login**, que consiste num ficheiro de configuração PAM-aware. Nesse ficheiro, o nosso grupo descomentou a seguinte linha:

```
# Uncomment and edit /etc/security/access.conf if you need to
# set access limits.
# (Replaces /etc/login.access file)
account  required       pam_access.so
```

Esta linha permite a adição do módulo pam_access.so ao serviço de login. Com ela, podemos editar o ficheiro **/etc/security/access.conf**, por forma a controlar como o módulo irá agir. No caso, o ficheiro contém as permissões do sistema.

Por forma a adicionarmos restrições para os vários utilizadores listados no ficheiro **/etc/bad-guys**, criámos o seguinte script, com o nome “denyAccessUsers”:

```
#!/bin/bash

n1=1
n2=1
n3=1
backupFile=/tmp/bad-guys.prev
actualFile=/etc/bad-guys
file=/etc/security/access.conf

while read line; do
if ! grep -q -s "$line" "$file"; then
        echo "-: $line : ALL" >> "$file"
fi
n1=$((n1+1))
done < "$actualFile"

while read linePrev; do
if ! grep -q -s "$linePrev" "$actualFile"; then
        sed -e "s/-: $linePrev : ALL//g" -i "$file"
fi
n2=$((n2+1))
done < "$backupFile"

while read actualFileLine; do
echo -n "" > "$backupFile"
echo "$actualFileLine" >> "$backupFile"
n3=$((n3+1))
done < "$actualFile"
```

Neste script, para cada utilizador presente no ficheiro **/etc/bad-guys**, verificamos se o mesmo já se encontra presente no ficheiro **/etc/security/access.conf**. Se o mesmo não se encontra lá, então será adicionada uma restrição de acesso ao sistema (no caso, para não conseguir aceder). Em caso contrário, nada acontece.

No entanto, um dos utilizadores definidos no ficheiro **/etc/bad-guys** pode ser eliminado do ficheiro. Por culpa disso, criamos um ficheiro backup que guarda sempre uma versão desse ficheiro quando o script é corrido (**/tmp/bad-guys.prev**).

Para verificarmos se algum elemento do ficheiro **/etc/bad-guys** foi eliminado, comparamos ambos (comparamos cada endereço do ficheiro backup com os endereços atuais do ficheiro original). Se o endereço não existir mais no original, eliminamos o mesmo do ficheiro **/etc/security/access.conf**.

Por forma a evitar ter que correr manualmente o script, editámos o ficheiro **/etc/crontab** para correr o script de minuto a minuto:

```
*  *    * * *   root    /etc/denyAccessUsers
```

Realização em Windows:

O nosso grupo decidiu não realização a parte deste User Story relativa à implementação do mesmo em Windows.

User Story 7:

Enunciado:

Como administrador da infraestrutura quero que as mensagens pré-login e pós-login bem sucedido sejam dinâmicas (por exemplo, “[Bom dia] | [Boa tarde] username”, etc.).

Realização em Linux:

Para a realização deste caso de uso, o nosso grupo decidiu criar 3 scripts, em que um deles possui uma mensagem de bom dia, outro uma mensagem de boa tarde e um último com uma mensagem de boa noite.

Todos os scripts supracitados apenas diferem na mensagem enviada para a consola, mensagem esta que é igual, tanto no pré como no pós login.

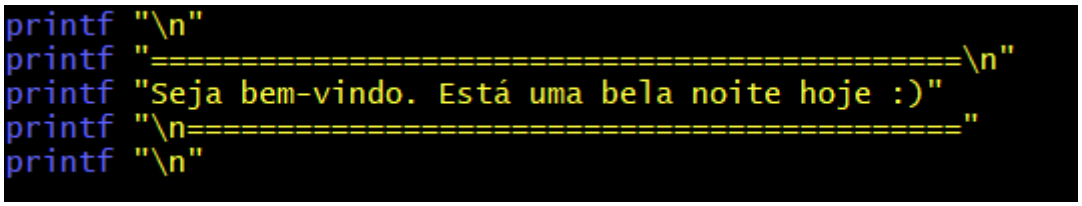
```
#!/bin/bash

preMessage='Bom dia, caro utilizador'
postMessage='printf "Seja bem-vindo. Está uma bela manhã hoje :)"'

sed -i "3s/.*/$preMessage/g" /etc/issue
sed -i "30s/.*/$postMessage/g" /etc/update-motd.d/00-header
```

Neste script, para o envio dinâmico de mensagens pré-login, o nosso grupo envia a mensagem presente na variável preMessage para o ficheiro **/etc/issue**, que é o ficheiro que contém uma mensagem que será imprimida aquando do pré-login na máquina. O grupo decidiu formatar o ficheiro da seguinte forma:

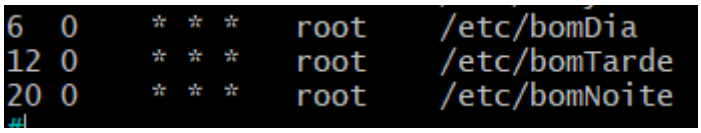
```
================================

Bom dia, caro utilizador

================================
```

Por forma a mostrar essa mesma mensagem quando são realizados acessos ao sistema via SSH, adicionámos um banner ao ficheiro **/etc/ssh/sshd_config**, que permite imprimir o conteúdo do ficheiro passado por parâmetro (no caso, **/etc/issue**) na consola aquando do login via SSH:

Para as mensagens pós-login, editámos o ficheiro **/etc/update-motd.d/00-header**, por forma a imprimir-mos a mensagem no header do motd, que é uma mensagem enviada para todos os utilizadores, quando os mesmos fazem login no sistema. O conteúdo passa a ser o seguinte:



Para que a mensagem alterna-se ao longo do dia, decidimos adicionar ao ficheiro **/etc/crontab** as seguintes tarefas:



De acordo com a imagem em cima, todos os dias, às 6 da manhã, o script bomDia é executado, às 12 horas, o script bomTarde é executado e, às 20 horas, o script bomNoite é executado, mudando, assim, as mensagens imprimidas no pré e pós logins.

Realização em Windows:

O nosso grupo decidiu não realização a parte deste User Story relativa à implementação do mesmo em Windows.

User Story 8:

Enunciado:

Como administrador da infraestrutura quero que o servidor Linux responda e envie pedidos ICMP para teste de conectividade apenas e só aos computadores dos elementos do grupo.

Realização:

Neste UC, o nosso grupo não conseguiu obter os IPs das máquinas de cada elemento. No entanto, caso tivéssemos conseguido obter, teríamos adicionado ao script com as configurações da firewall definido anteriormente 2 regras necessárias para que os pedidos ICMP pudessem ser feitos e recebidos. As regras seriam as seguintes:

Primeira regra => iptables -A INPUT -p icmp --icmp-type echo-request -d **IP**-j ACCEPT

Segunda regra => iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

A primeira regra apenas permite pacotes que transportam protocolo ICMP, para que o servidor consiga enviar pedidos ICMP, desde que o endereço de destino (-d) seja um **IP** correspondente

à máquina de um elemento do grupo. No caso, teríamos 5 regras, para 5 IPs diferentes, correspondentes a cada máquina dos elementos do grupo.

Já a segunda regra faz uso do módulo state (-m state). A opção -m permite carregar módulos, para suporte de funcionalidades adicionais. No caso, permite definir o critério de correspondência –state {estados}. Usariamos os estados ESTABLISHED e RELATED. O primeiro aplica-se a tráfego de pacotes TCP relativo a conexões estabelecidas previamente. Já o segundo, aplica-se a tráfego de pacotes UDP que sejam resposta a pedidos UDP anteriormente feitos.