Universidade do Minho
Departamento de Informática
Criptografia e Segurança em Redes **2023/2024 - Trabalho Prático 1**

Limite de submissão: 10/10/2023 - 23:59
Discussão do trabalho: 11/11/2023 - Durante o turno da aula prática.

Traffic sniffing & Packet spoofing

1 Instruções

O trabalho deverá ser desenvolvido em grupo de três integrantes e submetido exclusivamente via *blackboard* na data indicada acima. Submissões por outro canal serão desconsideradas. A partir da data e hora indicados, o grupo terá um desconto de 10% da classificação por dia de atraso na entrega, que não deverá ultrapassar o dia anterior a sessão de discusssão do respetivo trabalho.

Um único membro do grupo deverá submeter uma pasta compactada contendo:

- Relatório de trabalho com as respostas e discussões para cada uma das alíneas propostas ao longo do enunciado;
- Todo o código desenvolvido devidamente comentado.

A avaliação incluirá, além dos artefatos submetidos via *blackboard*, uma sessão de discussão onde todos os membros do grupo discutirão com o docente o trabalho produzido e servirá para a avaliação da contribuição individual. Nesta sessão, o grupo deverá demonstrar o ambiente funcional utilizado para dar resposta ao enunciado.

Note que apesar de o trabalho ser feito em grupo, a avaliação e classificação é individual.

2 Resumo e Objetivos

A análise de tráfego (*i.e., sniffing*) e falsificação (*i.e., spoofing*) de pacotes são dois conceitos importantes na segurança de rede e correspondem a ameaças significativas para a comunicação em rede. Conhecer detalhadamente estas duas ameaças é essencial para compreender as medidas de segurança em rede.

O *sniffing* de tráfego é uma técnica utilizada para monitorizar e capturar o tráfego de dados transmitido através de uma rede de computadores. Essa tarefa é geralmente realizada por administradores de rede (ou atacantes) e envolve a captura de pacotes de dados trafegando em uma rede para posterior análise. Isso pode ser feito por meio de software específico conhecido como *sniffer* ou analisador de protocolos, que permite que o utilizador visualize e interprete o conteúdo dos pacotes de dados. Existem diferentes ferramentas para *sniffing* de tráfego, *e.g.*, `Wireshark`, `Tcpdump`, `Scapy`, etc.

O *spoofing* de pacotes de redes é uma técnica na qual um atacante falsifica campos de um pacote de rede para mascarar sua verdadeira identidade ou localização. O tipo mais comum de *spoofing* de pacotes envolve a falsificação do endereço IP de origem de um pacote de rede com o objetivo de fazer com que o tráfego pareça vir de uma fonte legítima ou para ocultar a verdadeira origem do tráfego. `MAC` *spoofing* e `DNS` *spoofing* são, também, ataques comuns. Bibliotecas como o `libpcap` e `Scapy` são amplamente usadas em ações de *spoofing* de pacotes.

Apesar da importância de ser capaz de as utilizar, é fundamental entender como essas ferramentas funcionam e são implementadas. Assim, o objetivo deste trabalho prático consiste em aprender a utilizar e compreender as tecnologias subjacentes a estas ferramentas. Para isso, será proposto o desenvolvimento de programas simples que abragem os seguintes tópicos:

- Como funcionam programas para *sniffing* e *spoofing* de pacotes de redes;

- *Sniffing* de pacotes usando as bibliotecas `pcap` e `Scapy`;
- *Spoofing* de pacotes usando `raw socket` e `Scapy`;
- Manipulação de pacotes usando `Scapy`.

3 Ambiente de trabalho

O desenvolvimento do trabalho prático exigirá a instalação de um ambiente virtual, preferencialmente, baseado em um sistema operativo Linux. Para isso, instale no seu computador pessoal (*i.e.*, *Host* na Figura 1) um *software* de virtualização. As intruções deste enunciado são baseadas no `VMware`[1,2]. Contudo, poderão optar por outras soluções de vitualização como o `VirtualBox`[3] ou `UTM`[4].

Tendo o *software* de virtualização instalado, crie e instale uma VM Linux (*i.e.*, *Virtual Machine* na Figura 1) a sua escolha.

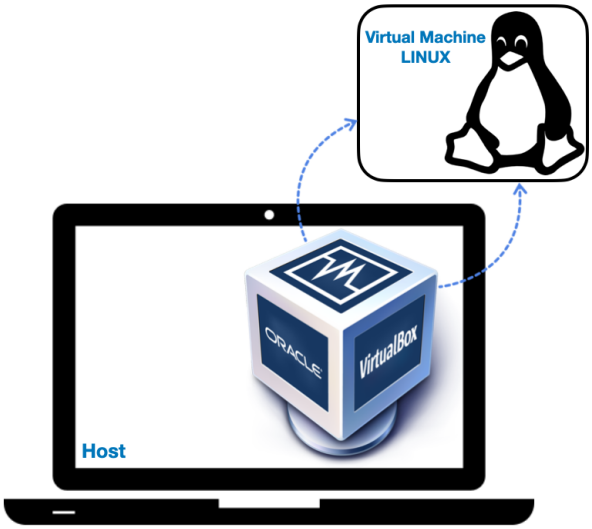

Figure 1: Arquitetura do ambiente de trabalho.

É importante que o adaptador de rede do ambiente virtual esteja configurado em modo *bridge*. Assim, o *Host* e a *VM* estarão conectados na mesma rede como dispositivos independentes. A Figura 2 mostra a opção que deve ser selecionada nas configurações do `VMware`.

Além do *software* de virtualização, deverá ter o `Wireshark`[5] instalado no sistema *host*. O `Wireshark` é uma ferramenta para análise de tráfego de rede de código aberto amplamente utilizada. Ele foi projetado para capturar, visualizar e analisar o tráfego de rede em tempo real ou a partir de ficheiros de captura previamente gravados em formato `pcap`. A documentação para a ferramenta está disponível neste link.

Será também no sistema *host* onde deverão correr os programas desenvolvidos pelo grupo e onde farão a demonstração do trabalho na sessão de discussão com o docente[6]. Para isso, deverão ter instalado o `Python 3.x` e a biblioteca `Scapy`[7]. A instalação da biblioteca `Scapy` pode ser feita através do terminal, usando o *Pip Installs Packages* (`pip`):

```
# pip install scapy
```

[1] https://www.vmware.com
[2] Adequado para sistemas Apple ARM64
[3] https://www.virtualbox.org
[4] https://mac.getutm.app
[5] https://www.wireshark.org
[6] Observe que as instruções neste enunciado são baseadas no sistema operativo Ubuntu Server 23.04
[7] A documentação para a biblioteca está disponível neste link.

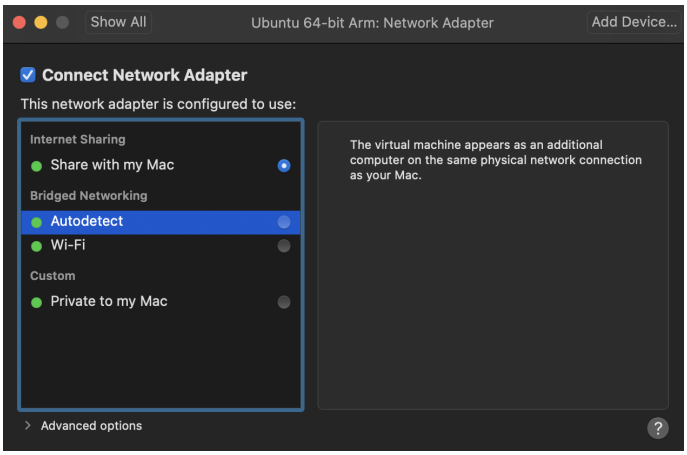

Figure 2: Configuração do adaptador de rede para a VM.

Caso não tenha o `pip` instalado, poderá obtê-lo usando o comando:

```
# sudo apt install python3-pip
```

É possível testar o `Scapy` usando o modo interativo do Python:

```
# python3
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
### [IP] ###
   version   = 4
   ihl       = None
   ...
```

4 Tarefa 1 - Análise de tráfego usando o `Wireshark`

O objetivo desta tarefa é a familiarização com as atividades de análise de tráfego. Para isso, será usado apenas o `Wireshark` instalado no sistema *host*. Certifique de executar a aplicação com permissões de administrador do sistema para que seja possível a captura de pacotes trafegados por sua interface de rede.

Em uma primeira fase, capture tráfego por um pequeno intervalo de tempo e analise a informação correspondente aos diferentes protocolos dos pacotes capturados. A Figura 3 exemplifica o sumário de um conjunto de pacotes. Use filtros para selecionar um subconjunto de pacotes de interesse, *e.g.,* com diferente protocolos de transporte (`TCP` e `UDP`), `ICMP`, `DNS`, etc. Analise os dados correspondentes a cada camada da *stack* de protocolos `TCP/IP`.

Após uma exploração inicial, inicie uma nova captura de tráfego e, em seguida, use o seu navegador para aceder as seguintes páginas da internet:

```
- www.scanme.org
- www.owasp.org
```

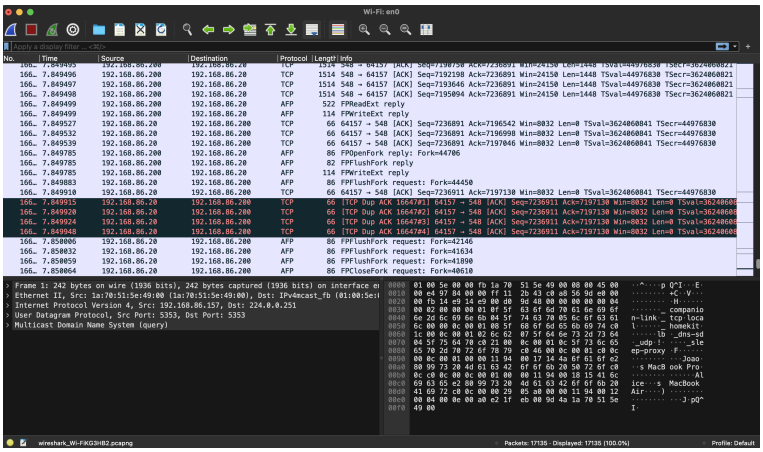

Figure 3: Exemplo de uma captura de tráfego usando o `Wireshark`.

Interrompa a captura de tráfego e use a informação correspondente para responder a Questão 1:

Questão 1

1.1 Quais as diferenças observadas nos protocolos usados em cada acesso?
1.2 De que maneira as diferenças observadas podem afetar a proteção das propriedades de segurança do tráfego em rede?
1.3 Use a opção *Analyze* → *Follow* do `Wireshark` para reconstruir o fluxo de cada acesso. Descreva o resultado e discuta como as propriedades de segurança são afetadas por cada protocolo usado na camada de aplicação.

5 Tarefa 2 - *Sniffing* de tráfico usando `Scapy`

O `Wireshark` é a ferramenta de *sniffing* de tráfego mais popular. No entanto, é difícil usá-la como um componente para construir outras ferramentas. Assim, objetivo desta tarefa é explorar a biblioteca `Scapy` para fazer *sniffing* em programas `Python`.

Os programas desenvolvidos nesta tarefa deverão ser executados no ambiente virtual descrito na Secção 3.

Um exemplo simples de uso da biblioteca `Scapy` através do modo interativo do Python é mostrado abaixo:

```
# python3

>>> from scapy.all import *

>>> def print_pkt(pkt):
>>>     pkt.show()

>>> pkt = sniff(iface='eth0', filter='icmp', prn=print_pkt)
```

O código acima irá capturar os pacotes na interface `eth0`. Se quisermos capturar tráfego em várias interfaces, podemos incluir todas as interfaces numa lista e atribuí-la a `iface`. Veja o exemplo a seguir:

```
iface = ['eth0','eth1','en0']
```

Ainda sobre o código exemplo, para cada pacote capturado, a função `print_pkt()` será invocada. Esta função apresentará algumas das informações sobre o pacote. Execute o programa e certifique que pode de facto capturar pacotes[8]. É possivel, também, armazenar os dados capturados em um fichero `pcap` para

[8]Caso execute o programa a partir de um ficheiro `.py`, deverá fazê-lo com privilégios de `root`

posterior análise, por exemplo, usando o `Wireshark`. Para isso, remova o parâmetro `prn=print_pkt` e adicione a seguinte linha ao seu programa `Python`:

```
wrpcap('/path/file\_name.pcap', pkt)
```

Tipicamente, quando capturamos tráfego, só estamos interessados em determinados tipos de pacotes ou protocolos. Podemos limitar a captura configurando filtros no *sniffing*. No exemplo acima, estamos a capturar apenas pacotes `ICMP` (ver linha 8). O filtro do `Scapy` usa a sintaxe `BPF` (Berkeley Packet Filter)[9].

Questão 2

Construa um *sniffer* baseado no `Scapy` e configure o filtro para capturar o tráfego correspondente às alíneas abaixo e que tenha como origem ou destino o sistema virtual (*VM* na Figura 1). Observe que deverá aplicar um filtro para cada alínea independentemente. Armazene as capturas em ficheiros `pcap` para posterior análise.

2.1 Capture apenas pacotes ARP[a];
2.2 Capture qualquer pacote TCP com um número de porta de destino 80. A partir da *VM*, volte a aceder as páginas da Questão 1 enquando executa o programa no sistema *host*. Compare os resultados das capturas provenientes da Questão 1 e 2.
2.3 Extenda o filtro para capturar todo o tráfego com origem ou destino na subrede do sistema virtual. Qual a diferença no resultado da captura?
2.4 Discuta como o `Scapy` poderia ser usado para violar a propriedade de segurança da disponibilidade (*i.e., availability*, discutida na aula teórica).

[a]Inclua no relatório capturas de imagem que demonstrem o resultado obtido.

6 *Spoofing* de pacotes `ICMP`

Como uma ferramenta de *spoofing* de tráfego, o `Scapy` permite-nos definir os campos dos pacotes IP com valores arbitrários. Com base nisso, o objetivo desta tarefa é falsificar pacotes ICMP[10] (*i.e.,* `ICMP echo request`). Além disso, usaremos o `Wireshark` para observar o tráfego resultante. Caso o pedido seja aceite pelo recetor, será enviado um pacote de resposta (*i.e.,* `ICMP echo reply`) para o endereço IP forjado.

O código abaixo mostra um exemplo de *spoofing* de um pacote ICMP.

```
1  # python3
2
3  >>> from scapy.all import *
4
5  >>> a = IP()
6  >>> a.dst = '192.168.1.1'
7  >>> b = ICMP()
8  >>> p = a/b
9  >>> send(p)
10 .
11 Sent 1 packets.
```

No código acima, a *linha 5* cria um objeto `IP` a partir da classe `IP`; A *linha 6* mostra como definir o campo do endereço IP de destino. Se um campo não for definido, será utilizado um valor predefinido; A *linha 7* cria um objeto ICMP. O tipo predefinido é `echo request`; O operador / é redefinido pela classe IP, pelo que já não representa a divisão. Em vez disso, significa adicionar `b` como o campo do *payload* de `a` e modificar os campos de `a` correspondentes. Como resultado, obtemos um novo objeto que representa um pacote `ICMP`; Podemos agora enviar este pacote usando `send()` na *linha 9*.

[9]Pode consultar um material introdutório neste link.
[10]Internet Control Message Protocol

Questão 3

Construa um programa capaz de fazer *spoofing* de pacotes `ICMP`. Configure o seu programa para enviar o `echo request` para o endereço `IP` 8.8.8.8. Configure o endereço de origem com o `IP` atribuído ao seu ambiente virtual. Execute o programa no sistema *host* tendo o `Wireshark` a capturar tráfego da *interface* ligada a mesma rede da *VM*.

3.1 Analise a troca de pacotes correspondentes ao tráfego gerado. Discuta os resultados obtidos no teste[a].
3.2 Quais propriedades de segurança são violadas pelo programa criado nesta tarefa?
3.3 Use um analisador de tráfego (`Wirehark` ou o seu programa criado na Tarefa 2) no ambiente virtual para observar se o pacote `echo reply` é entrege neste sistema[a].
3.4 Adapte o seu programa para que seja capaz de fazer *spoofing* de pacotes `ICMP` enviados pelo sistema virtual do seu ambiente de trabalho[b].

[a] Inclua imagens no relatório que suportem a sua discussão.
[b] Inclua no relatório a descrição da adaptação feita ao programa original.