**Pontifícia Universidade Católica do Rio Grande do Sul**

**Laboratório de Redes de Computadores**

**Engenharia de Software**

**Carolina Ferreira, Felipe Freitas, Luiza Heller e Mateus Caçabuena**

**Trabalho 2**

**Porto Alegre**

**2024**

Sumário

[**1. Introdução 3**](#_Toc183464294)

[**2. Descoberta de Hosts 4**](#_Toc183464295)

[**2.1. Funcionamento do Código 4**](#_Toc183464296)

[**3. Topologia 9**](#_Toc183464297)

[**3.1. Topologia Inicial 9**](#_Toc183464298)

[**3.2. Topologia Simulada (Pós-ataque) 9**](#_Toc183464299)

[**4. Ataque 10**](#_Toc183464300)

[**4.1. Execução de Ataque 10**](#_Toc183464301)

[**4.2. Checagem de Ataque 14**](#_Toc183464302)

[**5. Monitoramento de Tráfego 15**](#_Toc183464303)

[**6. Conclusão 20**](#_Toc183464304)

# Introdução

O presente relatório descreve o desenvolvimento e a execução de um projeto cujo objetivo foi implementar um ataque do tipo *man-in-the-middle* para capturar o histórico de navegação web de um computador alvo em uma rede local. Este trabalho foi conduzido no contexto da disciplina de Redes, da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), e teve como base a exploração de vulnerabilidades em redes utilizando técnicas avançadas de monitoramento e interceptação de tráfego.

A proposta foi dividida em três etapas principais: a descoberta de hosts ativos na rede, a execução do ataque de ARP Spoofing e a análise do tráfego capturado. Inicialmente, desenvolvemos uma aplicação para identificar os dispositivos conectados à rede, utilizando mensagens ICMP para determinar a atividade dos hosts. Na sequência, realizamos um ataque *man-in-the-middle* por meio de ARP Spoofing, permitindo a interceptação do tráfego entre o alvo e o roteador. Por fim, implementamos uma aplicação para capturar e analisar pacotes DNS e HTTP, reconstruindo o histórico de navegação do host alvo.

Este relatório apresenta uma descrição detalhada de cada etapa, incluindo os métodos e ferramentas utilizadas, os desafios enfrentados e os resultados obtidos. Além disso, são apresentados testes e análises realizados com o auxílio da ferramenta Wireshark, evidenciando a eficácia da solução desenvolvida.

# Descoberta de Hosts

A etapa de descoberta de hosts teve como objetivo identificar os dispositivos ativos em uma rede local, replicando o comportamento de uma varredura inicial no estilo ping scan. O código do arquivo host\_discovery.py foi utilizado para realizar essa descoberta.

## 2.1. Funcionamento do Código

O programa utiliza **socket raw** para criar pacotes ICMP personalizados, contendo cabeçalhos IP e ICMP. As principais funcionalidades do código incluem:

**1. Estruturas de Cabeçalho**

O código define as classes IP e ICMP para criar e manipular os cabeçalhos dos pacotes:

* **IP Header**:
  + Contém informações como versão (IPv4), TTL (*Time to Live*), e endereços de origem e destino.
  + Criado utilizando a função socket.inet\_aton para converter endereços IP em um formato binário.
* **ICMP Header**:
  + Contém o tipo (8 = requisição, 0 = resposta), código, checksum, identificador e número de sequência.
  + O checksum é calculado pela função calculate\_checksum, garantindo a integridade do pacote.

**2. Criação de Pacotes**

A função create\_packet combina os cabeçalhos IP e ICMP em um único pacote. Ela realiza:

* Montagem do cabeçalho IP usando struct.pack.
* Geração do checksum do pacote ICMP.
* Reempacotamento do cabeçalho ICMP com o checksum atualizado.

**3. Envio de Pacotes**

A função scan\_host realiza a varredura de um host específico:

* Cria um *socket* bruto e envia o pacote ICMP.
* Aguarda a resposta (usando recvfrom), registrando o tempo de resposta.
* Caso o host não responda dentro do tempo limite (timeout), ele é considerado inativo.

**4. Varredura da Rede**

A função scan executa a varredura em múltiplos hosts:

* Cria threads para realizar a varredura paralela de cada endereço IP da rede especificada.
* Filtra os endereços de *broadcast* e da rede.
* Ordena e exibe os resultados, destacando o pior tempo de resposta e a quantidade de hosts ativos.

**5. Saída do Programa**

Ao final da varredura, o programa exibe:

* Lista de hosts ativos e seus tempos de resposta.
* Total de hosts ativos, inativos e na rede.
* Pior tempo de resposta registrado.

A imagem fornecida no relatório do Wireshark demonstra os resultados práticos da varredura. Foram capturadas mensagens ICMP (protocolo *ping*), enviadas para diferentes hosts da rede, com respostas indicando os dispositivos ativos. Para cada mensagem de requisição (*request*) enviada, o Wireshark registrou uma resposta (*reply*), validando a presença de um host ativo naquele endereço IP.

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Na captura anexada, é possível observar:

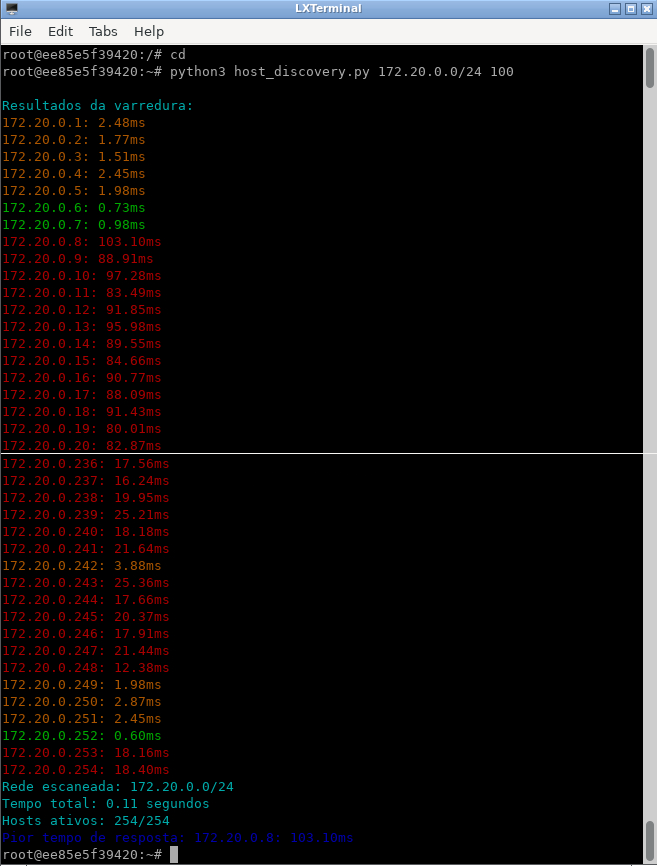
* **Protocolo Utilizado**: Mensagens ICMP (ping).
* **Atividade Registrada**: Os endereços IP das fontes e destinos indicam os hosts na rede sendo pingados (src: 192.168.15.64 para diferentes destinos na rede 192.168.15.X).
* **Respostas ICMP**: Para cada requisição, a linha correspondente mostra "reply in X ms", indicando que o host respondeu ao *ping* com o tempo de resposta registrado.
* **Hosts Ativos e Inativos**: O log também contém mensagens como "No response found!", indicando hosts que não responderam ao ping, ou seja, considerados inativos.

Esses dados validam a funcionalidade do script, que foi capaz de listar os IPs ativos conforme o objetivo do trabalho, utilizando uma abordagem sistemática baseada em ICMP.

Para verificar quais hosts estão ativos na rede, você pode executar o seguinte comando em qualquer um dos contêineres:

$ python3 host\_discovery.py 172.20.0.0/24 100

Fazendo isso, retorna o que foi retratado na imagem a seguir:



A figura 2 retrata saídas similares as do Wireshark nas quais podemos ver algumas respostas de outras máquinas presentes na rede e seus tempos de resposta em milisegundos. Em verde, respostas muito rápidas (menor ou igual à 1ms), em amarelo respostas aceitáveis (até 10ms) e em vermelho as demais respostas que demandam maior quantia de tempo.

# Topologia

## 3.1. Topologia Inicial

Assim estão dispostos os 3 hosts pela rede. O gateway é um nodo central que está conectado à internet e aos 3 containeres. Seu IP é 172.20.0.254/24:

A diagram of a network

Description automatically generated

## 3.2. Topologia Simulada (Pós-ataque)

O atacante se posicionaria no meio da topologia servindo de ponte obrigatória para todos os componentes envolvidos:

A diagram of a network

Description automatically generated

# Ataque

Após a identificação dos hosts alvos na rede, esta etapa foca na execução de um ataque do tipo ARP Spoofing para estabelecer uma posição de *man-in-the-middle* (MITM) entre o dispositivo alvo e o roteador da rede e dois dispositivos alvos. O objetivo deste ataque é manipular as tabelas ARP dos dispositivos, redirecionando o tráfego de rede através do atacante, permitindo a interceptação e monitoramento das comunicações entre o host alvo e outros dispositivos.

Antes de iniciar o ataque, vamos gerar tráfego para ver a tabela ARP ideal (com os endereços MAC apropriados):

## 4.1. Execução de Ataque

**Primeira Vítima**

**# Ping other victim (to generate traffic)**

**$ ping 172.20.0.3 -c 3**

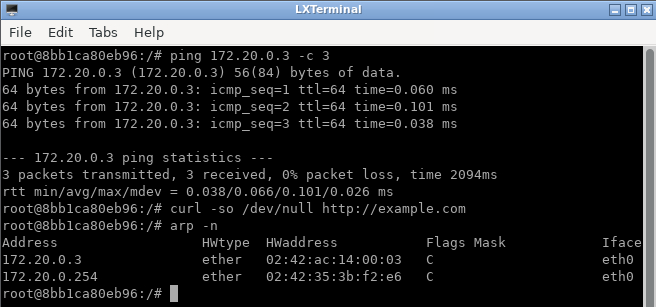
**# Send a request to the outside through the gateway (to generate traffic)**

**$ curl -so /dev/null http://example.com**

**# Check the ARP table**

**$ arp -n**

Executando estes commandos, retorna o que é retratado na imagem abaixo:



**Segunda Vítima**

**# Ping other victim (to generate traffic)**

**$ ping 172.20.0.2 -c 3**

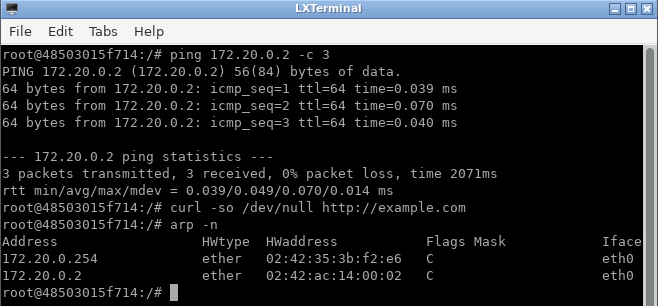
**# Send a request to the outside through the gateway (to generate traffic)**

**$ curl -so /dev/null http://example.com**

**# Check the ARP table**

**$ arp -n**

Executando estes commandos, retorna o que é retratado na imagem abaixo:



Observando a figura 3 e 4 acima, os pings foram resolvidos com sucesso para outro host. Em seguida, conclui-se que a requisição para um site externo (passando pelo gateway) foi um sucesso. Por fim, apresentamos a tabela ARP da primeira vítima com os endereços MAC corretos de cada máquina (vítima vizinha e gateway, respectivamente)

**Atacando a Rede**

No host atacante, é aberto 3 terminais para executar os ataques:

**# Attack victim 1 and gateway connection**

**$ ./arpspoof.sh 172.20.0.2 172.20.0.254**

**# Attack victim 2 and gateway connection**

**$ ./arpspoof.sh 172.20.0.3 172.20.0.254**

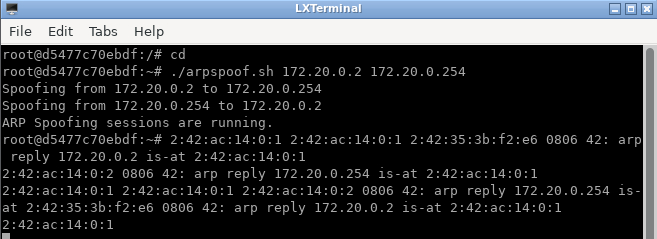
**# Attack victim 1 and victim 2 connection**

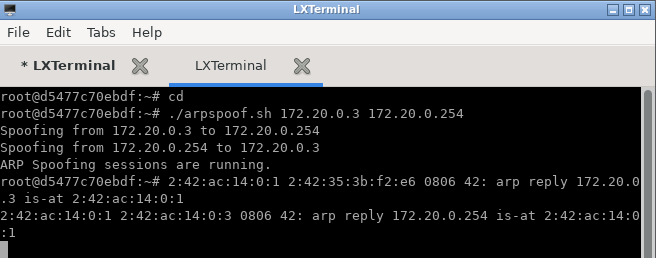
**$ ./arpspoof.sh 172.20.0.2 172.20.0.3**

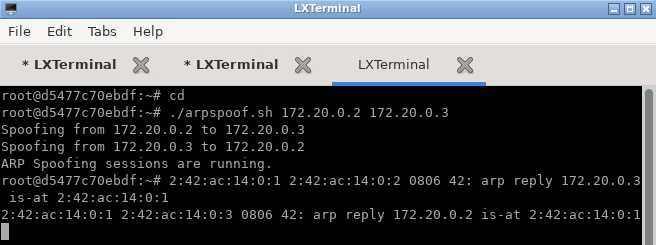
Executando estes comandos, é “inserido” o host entre a primeira vítima e o gateway, entre segunda vítima e o gateway e entre primeira vítima e segunda vítima, respectivamente.

Utilizamos um script customizado para rodar o ARP spoofing, para evitar de ter que rodar dois comandos para cada ataque. O script em questão roda a própria ferramenta arpspoofing local com os dois IPs passados por parâmetro, cada vez alterando a origem e o destino.

Nas 3 figuras abaixo, podemos ver a saída do program a ARP spoofing enviando o ARP reply para as vítimas:







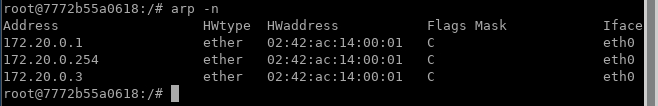
Agora, é necessário conferir que o ataque foi um sucesso acessando as duas vitimas.

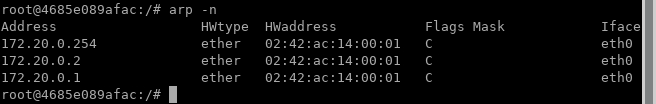
## 4.2. Checagem de Ataque

**# Check the ARP table**

**$ arp -n**

Agora a tabela ARP deve ter o endereço MAC do invasor associado ao endereço IP do gateway e da outra vítima, retornando o conteúdo da imagens a seguir:





# Monitoramento de Tráfego

Neste tópico, será abordado o processo de monitoramento do tráfego de navegação web das vítimas, utilizando técnicas de captura de pacotes para análise detalhada. O objetivo é desenvolver uma aplicação capaz de interceptar e inspecionar pacotes HTTP e DNS trafegados pela rede, permitindo rastrear o histórico de navegação do host. Essa etapa é essencial para compreender os padrões de comunicação do alvo, identificando solicitações de nomes de domínio (DNS) e acessos a páginas web (HTTP). O monitoramento será realizado com ferramentas e técnicas apropriadas para capturar o tráfego de forma eficiente, preservando a integridade do fluxo original de dados entre o host e o gateway.

Para monitorar o tráfego, é executado o seguinte comando para o atacante:

**$ python3 traffic\_sniffer.py**

O código utilizado (traffic\_sniffer.py) implementa um analisador de pacotes de rede que captura e processa pacotes em uma interface de rede especificada (eth0). Ele realiza as seguintes funções principais:

1. **Criação de Socket de Rede**: Configura um **socket raw** que opera no nível da camada de enlace, capturando todos os pacotes transmitidos na interface de rede especificada.
2. **Análise de Cabeçalhos**:
   * **Cabeçalho Ethernet**: Extrai informações da camada de enlace, identificando se o pacote transporta dados IP.
   * **Cabeçalho IP**: Extrai informações como o protocolo de transporte (TCP/UDP/ICMP) e os endereços IP de origem e destino.
   * **Cabeçalhos TCP e UDP**: Processa os dados adicionais das camadas de transporte, como portas de origem e destino.
   * **Pacotes DNS**: Analisa pacotes DNS para extrair os domínios consultados.
   * **Requisições HTTP**: Analisa pacotes HTTP para capturar o host e a URL acessada.
3. **Monitoramento de Tráfego**: O código identifica pacotes HTTP (porta 80) e DNS (porta 53), extraindo informações úteis como domínios consultados, URLs acessadas e o protocolo utilizado.
4. **Armazenamento de Histórico**: Os dados capturados (timestamp, hostname, protocolo, domínio e URL) são armazenados em uma lista e, ao final, exportados para um arquivo HTML formatado como uma tabela.
5. **Configuração de Parâmetros**:
   * A interface de rede, o número máximo de pacotes a capturar, o tempo limite de captura e o arquivo de saída podem ser configurados via argumentos da linha de comando. Estas entradas são opcionais e úteis para debugs. O padrão dos parametros é:
     1. interface=eth0
     2. packet\_limit=Infinity
     3. time\_limit=Infinity
     4. output\_file=output/history.html
6. **Execução**:
   * O programa captura os pacotes em tempo real até atingir o limite de tempo ou pacotes (ou ser interrompido manualmente).
   * No encerramento, salva o histórico em um arquivo HTML que pode ser utilizado para análise posterior.

Este programa é útil para monitorar tráfego de rede de forma passiva, permitindo capturar e analisar atividades relacionadas à navegação web e consultas de nomes de domínio.

**Geração de Tráfego**

Agora, temos que simular tráfego sendo gerado pelas vítimas, para que seja interceptado pelo atacante. Na primeira vítima, rodamos o seguinte script:

$ ./generate-http-traffic.sh

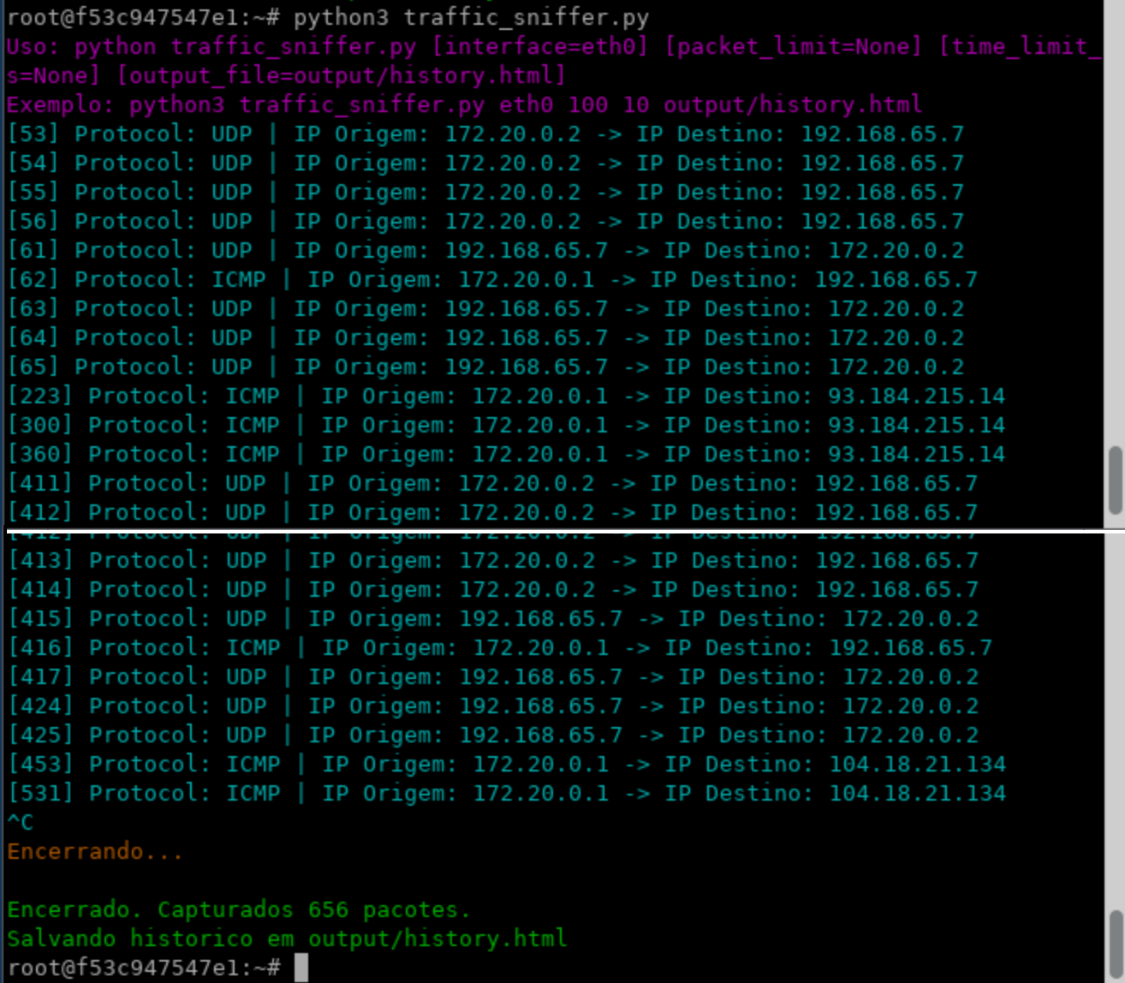
Este código em *bash script* é projetado para gerar tráfego HTTP na rede, enviando requisições para uma lista de URLs específicas usando o comando curl. Abaixo está a explicação detalhada de cada parte do script:

1. **Definição de URLs HTTP**:
   * O script armazena uma lista de URLs (todas com protocolo HTTP) em um array chamado urls.
   * Exemplos de URLs foram baseadas nas disponibilizadas no enunciado.
2. **Geração de Tráfego HTTP**:
   * O *loop* for itera sobre cada URL na lista.
   * Para cada URL:
     + Exibe uma mensagem indicando qual URL está sendo requisitada com echo "Requesting: $url".
     + Utiliza o comando curl para enviar uma requisição HTTP para o servidor correspondente. O parâmetro -s silencia a saída do comando, e -o /dev/null descarta qualquer conteúdo retornado pelo servidor.
     + Introduz uma pausa de 1 segundo entre cada requisição com o comando sleep 1, simulando comportamento humano ou evitando excesso de tráfego em um curto intervalo de tempo.

Na vítima 2 será gerado o tráfego HTTPS, para isso, basta acessar o browser e acessar alguns sites.

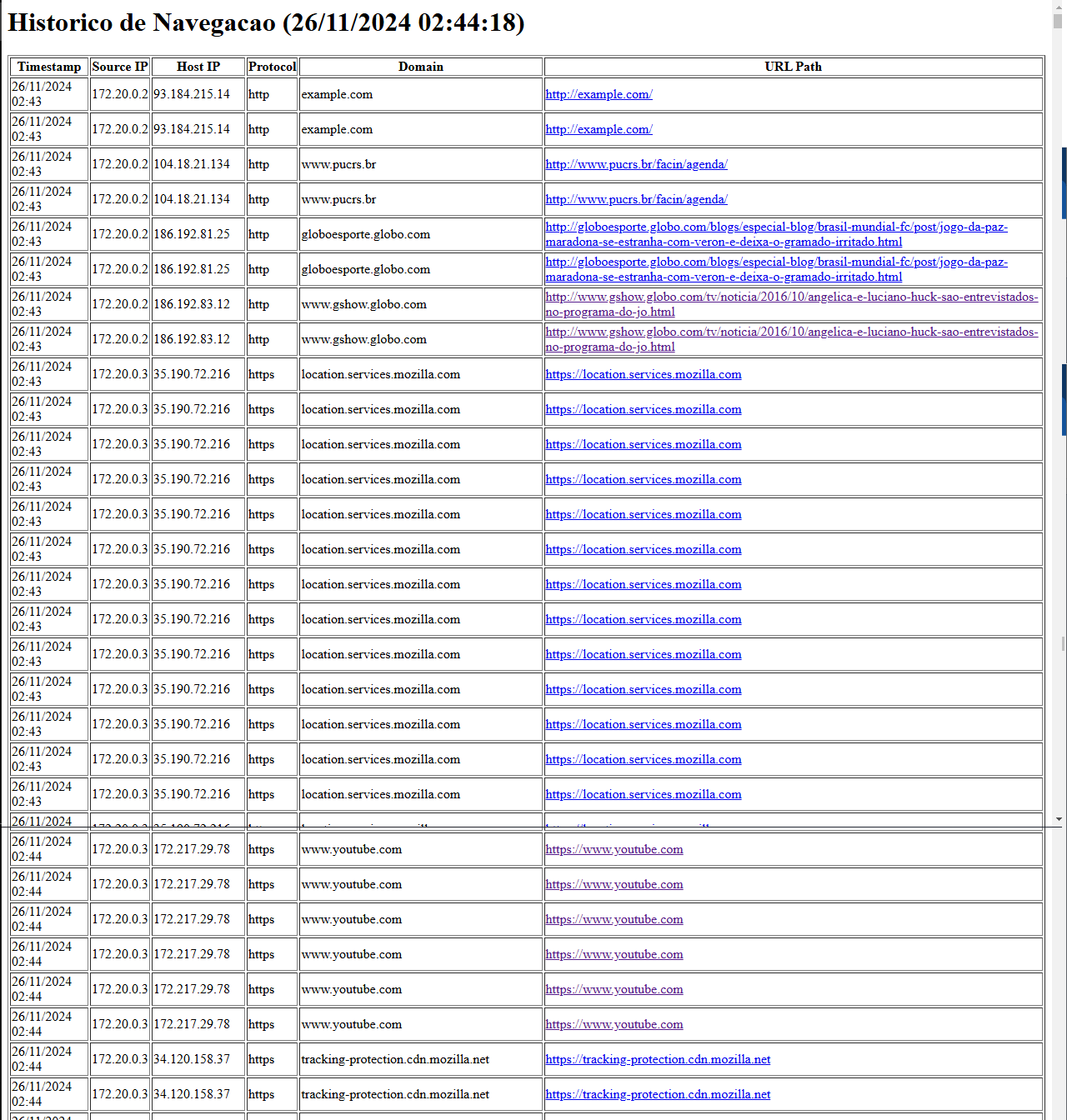
**Máquina Atacante**

Na foto a seguir, podemos ver alguns pacotes capturados



Essas imagens retratam a origem e destino de cada requisição (que agora está passando pelo container atacante).

Ao finalizar o programa, é utilizado um volume para mapear este arquivo de saída para a máquina local que rodou os containeres docker. Por ser um arquivo html, pode ser acessado de maneira mais estética através de qualquer navegador.



Arquivo html escrito após análise do trafego, as URLs http foram capturadas com sucesso e podemos ver claramente requisições tanto HTTP quanto HTTPS sendo registradas e qual sua origem. No caso, temos a vítima 1 (host 172.0.0.2) fazendo requisições HTTP e a vítima 2 (host 172.20.0.3) fazendo requisições HTTPS, como acessando o Youtube por exemplo.

# Conclusão

Este trabalho demonstrou, por meio de uma abordagem prática, como explorar vulnerabilidades em redes locais para implementar um ataque man-in-the-middle utilizando técnicas de ARP Spoofing. A partir de uma estrutura organizada em três etapas principais – descoberta de hosts, execução do ataque e monitoramento de tráfego –, foi possível compreender e aplicar conceitos avançados de redes, segurança e análise de tráfego.

Na etapa de descoberta de hosts, a aplicação desenvolvida identificou dispositivos ativos na rede por meio de mensagens ICMP personalizadas, validando a eficácia do mapeamento com capturas no Wireshark. Em seguida, o ataque ARP Spoofing manipulou as tabelas ARP do host alvo e do gateway, inserindo o atacante no fluxo de comunicação e redirecionando pacotes com sucesso. Por fim, o monitoramento de tráfego implementado capturou e analisou pacotes DNS e HTTP, reconstruindo o histórico de navegação em um formato HTML intuitivo.

Os resultados obtidos demonstram o impacto significativo que vulnerabilidades como a falta de proteção contra ARP Spoofing podem ter em redes locais. Ao mesmo tempo, o trabalho reforça a importância de implementar medidas de segurança para mitigar ataques desse tipo.

Além de alcançar os objetivos propostos, o projeto contribuiu para o aprofundamento de conhecimentos práticos em redes e segurança, evidenciando a relevância de explorar cenários reais para consolidar a aprendizagem teórica.