



Engenharia de Software
Laboratório de Redes de Computadores

Relatório Trabalho Final

DHCP Spoofing

Alunos: Aléxia Dorneles Pereira e Gustavo G. Lottermann
Professor: Sergio Johann Filho
Porto Alegre, 01 de julho de 2024

Introdução

Esse trabalho consiste na criação/implementação de um programa *Python* utilizando *sockets RAW* para o envio e recepção de pacotes DHCP no protocolo IPv4. Ele teve como objetivo principal simular um ataque do tipo *man-in-the-middle* através do envio de mensagens DHCP forjadas, permitindo o controle sobre a rota *default* dos *hosts* na rede local e a resolução de endereços no DNS.

Com isso, nosso código tem como objetivo desenvolver um programa em *socket RAW* para manipulação de pacotes DHCP, implementar a detecção e respostas às mensagens principais de DHCP que são: *Discovery* e *Request*, enviar mensagens DHCP para configurar os clientes (*Offer* e *Acknowledgement*) e, por final, simular um ataque *man-in-the-middle* para controlar a rota *default* e a resolução DNS dos *hosts* na rede.

Implementação

Para a implementação do nosso trabalho, foram criados 3 arquivos, arquivo *dhcp.py*, *payloads.py* e *sniffer.py*.

sniffer.py:

Neste arquivos criamos a função ***main*** e várias outras funções. A função *main* é a função principal, na qual ela captura pacotes de rede em tempo real, utilizando *sockets RAW*, ela identifica pacotes DHCP, extrai as informações necessárias e responde com pacotes DHCP modificados (*Offer* e *Acknowledgment*) para configurar os clientes com os parâmetros de rede controlados pelo controlador da rede. Fizemos ela funcionando da seguinte forma: ela cria os *sockets* para a captura e envio de pacotes, captura pacotes de rede e analisa os cabeçalhos *Ethernet* e IPv4, identifica pacotes TCP, UDP e DHCP, extraíndo as informações importantes desses pacotes e por final responde pacotes DHCP *Discovery* com pacotes DHCP *Offer* e pacotes DHCP *Request* com pacotes DHCP *Ack*.

Já para o caso das funções auxiliares:

- ***parse_ether_header(raw_data):*** Analisa o cabeçalho Ethernet.

- **`parse_ipv4_header(raw_data)`**: Analisa o cabeçalho IPv4.
- **`parse_tcp_header(raw_data)`**: Analisa o cabeçalho TCP.
- **`parse_udp_header(raw_data)`**: Analisa o cabeçalho UDP.

dhcp.py:

Nesse arquivo foram criadas as classes ***DCHPHandler*** e ***DCHPProtocol***, na qual a classe *DCHPHandler* é responsável por analisar e extrair as informações dos pacotes DHCP recebidos, assim ela extrai e converte os dados brutos dos pacotes em informações úteis, como endereços IP e de hardware, e várias opções de DHCP. Já a classe *DCHPProtocol* define constantes para portas e códigos de opções DHCP, além de fornecer métodos para obter o tipo de mensagem DHCP.

Os principais métodos dessas classes são:

- **`extract_data()`**: extrai e analisa dados do payload DHCP. Extrai o endereço IP do cliente e o endereço de hardware do cliente dos dados brutos;
- **`extract_options()`**: analisa e extrai opções DHCP do *payload*, a exemplo da lista de solicitações, tipo de mensagem, nome do *host*, IP solicitado e ID do servidor;
- **`get_message_type(value)`**: retorna o tipo de mensagem DHCP dependendo do valor passado por parâmetro.

payloads.py:

Neste arquivo estão presentes a classe ***DHCPFrame*** e o enum ***DHCPOptions***. O primeiro responsável pela construção da carga dos pacotes DHCP, onde é convertido os campos de configuração DHCP em um formato de bytes que é adequado para enviar através da rede e o segundo é onde são definidos os métodos estáticos que geram as opções específicas para pacotes DHCP *Offer* e *Acknowledgement*.

Alguns métodos principais desse arquivo são:

- **`to_bytes()`**: converte *DHCPFrame* em *bytes* para envio;
- **`get_options_by_type(option_type)`**: retorna as opções de DHCP de acordo com o tipo de pacote (OFFER ou ACK);

- `get_offer_options()`: retorna as opções para o pacote DHCP OFFER.
- `get_ack_options()`: retorna as opções para o pacote DHCP ACK.

Configuração do Servidor DNS

Com nosso código pronto, fizemos a configuração do servidor DNS (BIND9) para redirecionar consultas DNS de sites conhecidos, no nosso caso utilizamos o www.google.com, para endereços IP controlados e definidos pelo controlador da rede. Dessa forma, utilizamos o servidor DNS BIND9, que nos permitiu definir zonas DNS que redirecionam consultas/pesquisas para IPs específicos já definidos.

Para que isso fosse feito, instalamos o pacote BIND9 no servidor que desejamos configurar ele. Após isso, adicionamos a seguinte configuração da zona para o domínio "google.com":

```
zone "google.com" {  
    type master;  
    file "/etc/bind/db.google.com";  
};
```

E por final, adicionamos os registros DNS no arquivo da zona dessa forma:

```
labredes@labredes:~$ cat /etc/bind/db.google.com  
;  
; BIND data file for local loopback interface  
;  
  
$TTL    604800  
@       IN      SOA     localhost. root.localhost. (  
                          1           ; Serial  
                      604800        ; Refresh  
                     86400         ; Retry  
                    2419200        ; Expire  
                   604800 )       ; Negative Cache TTL  
;  
@       IN      NS      localhost.  
@       IN      A       201.54.156.25  
www    IN      A       201.54.156.25
```

Assim, associamos o nome "www.google.com" ao endereço IP 201.54.156.25, que corresponde ao endereço IP do moodle da PUCRS. Com isso feito, reiniciamos o serviço BIND9 para aplicar as mudanças com o comando `sudo systemctl restart bind9`. Dessa forma, conseguimos executar um ataque *man-in-the-middle*, pois conseguimos fazer com que o controlador monitore e manipule o tráfego da rede, redirecionando as consultas DNS para o destino desejado, como podemos constatar no pacote 289 do protocolo DNS (imagem abaixo), na qual uma consulta DNS foi enviada para o site do *google*, porém o servidor BIND9 respondeu com o endereço IP configurado 201.54.156.25.

