Implementação do suporte à pacotes IPv6 e multiplataforma do Python Packet Sniffer

Lucas Martins de Barros

Link do repositório original: https://github.com/buckyroberts/Python-Packet-Sniffer
Link do fork: https://github.com/lucasmartinsb/lmb-Python-Packet-Sniffer

1. Introdução

Este trabalho teve como objetivo adicionar o suporte à leitura de pacotes IPv6 a um projeto já existente de Sniffer de pacotes Python. Além do suporte ao IPv6, o código também foi refatorado para se tornar multiplataforma, já que, originalmente, o programa funcionava somente em sistemas operacionais Linux.

O programa inicializa capturando pacotes que trafegam pela rede. Cada pacote capturado é decomposto em uma classe Ethernet, que extrai os dados de endereço MAC de destino, origem e protocolo.

Inicialmente, o programa verificava se o protocolo era IPv4, e caso não fosse, exibia os dados brutos. Se o pacote era IPv4, era realizada a identificação do tipo de cabeçalho, ICMP, UDP ou TCP e exibia os dados de acordo com o protocolo. O objetivo deste trabalho é implementar a mesma regra utilizada em pacotes IPv4 para pacotes IPv6, exibindo os dados de cabeçalho do pacote, assim como os dados específicos de protocolo. Além da análise dos pacotes, também foi alterada a forma com que capturava os pacotes, utilizando Scapy, que permitiu o programa se tornar multiplataforma.

2. Uso do Scapy

O programa, inicialmente fazia uso da biblioteca "socket", que realizava a captura de pacotes utilizando o parâmetro "socket.AF_PACKET". Este parâmetro, entretanto, é específico do Linux, o que impedia a execução em outros sistemas operacionais, como o Windows e MacOS.

Para resolver essa limitação, a biblioteca "socket" foi substituída pela "Scapy", outra ferramenta de manipulação de pacotes, que abstrai as diferenças entre sistemas operacionais. A nova biblioteca também muda a forma com que os pacotes são manipulados, que trata de uma forma mais simples do que a socket. Um exemplo de alteração é na classe UDP:

```python
class UDP:
    def __init__(self, raw_data):
        self.src_port, self.dest_port, self.size = struct.unpack('! H H 2x H', raw_data[:8])
        self.data = raw_data[8:]
```

Figura 1 - Classe UDP com Socket

```python
class UDP:
    def __init__(self, packet):
        self.src_port = packet.sport
        self.dest_port = packet.dport
        self.size = packet.len
        self.data = bytes(packet.payload)
```

Figura 2 - Classe UDP com Scapy

As Figuras 1 e 2 mostram a mesma classe, que manipula um pacote UDP com um pacote recebido pela biblioteca sockets e outro pela Scapy. A biblioteca Scapy tem um alto nível de abstração, fornecendo acesso diretamente à campos dos pacotes sem a necessidade de entender o layout dos cabeçalhos do pacote.

Com a biblioteca sockets é necessário extrair manualmente os dados do pacote, pelo formato do cabeçalho. Como no exemplo da Figura 1, para extrair os dados de portas origem, destino e tamanho é necessário ler os primeiros 8 bytes dos "dados crus", como dois números de 16 bits ("H") para as portas origem e destino, ignorando 2 bytes ("2x") e interpretando o comprimento como outro número de 16 bits ("H"). Enquanto isso, o Scapy, por baixo dos panos, já realizou essas operações e armazenou nos atributos "sport", "dport" e "len" do pacote.

Para utilizar o Scapy foi necessário refatorar todas as classes, mudando a forma de manipular os pacotes, para uma forma semelhante à Figura 2. Por outro lado, todos os atributos das classes foram mantidos como o original, para não afetar a exibição destas informações.

3. Pacotes IPv6

Para suportar a leitura de pacotes IPv6 foi criada uma nova classe, chamada IPv6 (na pasta networking, seguindo o mesmo padrão do autor original). O cabeçalho do pacote IPv6 é decomposto em: versão, classe de tráfego, rótulo de fluxo, tamanho dos dados, próximo cabeçalho, limite de encaminhamento, origem, destino e dados.

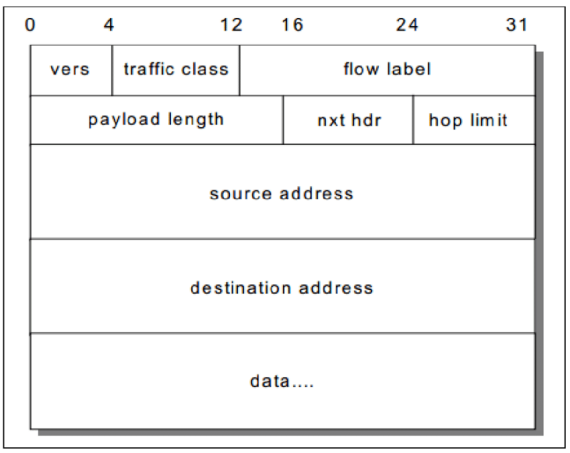

Figura 3 - cabeçalho IPv6

Além do ICMP, todas as outras classes do IPv4 funcionam da mesma maneira para o IPv6. O ICMP possui outro nome de camada quando é IPv6, é chamado de "ICMPv6ND_NS" pelo Scapy. Portanto, o teste se a camada existe no processamento do pacote é feito de uma forma diferente do ICMP do IPv4:

```
if 'ICMP' in packet:
    icmp = ICMP(packet['ICMP'])
    print(TAB_1 + 'ICMP Packet:')
    print(TAB_2 + 'Type: {}, Code: {}, Checksum: {},'.format(icmp.type, icmp.code, icmp.checksum))
    print(TAB_2 + 'ICMP Data:')
    print(format_multi_line(DATA_TAB_3, icmp.data))

elif 'ICMPv6ND_NS' in packet:
    icmpv6 = ICMPv6(packet['ICMPv6ND_NS'])
    print(TAB_1 + 'ICMP Packet:')
    print(TAB_2 + 'Type: {}, Code: {}, Checksum: {},'.format(icmpv6.type, icmpv6.code, icmpv6.checksum))
    print(TAB_2 + 'ICMP Data:')
    print(format_multi_line(DATA_TAB_3, icmpv6.data))
```

Figura 4 - comparação ICMP IPv4 e IPv6

Por outro lado, os campos de cabeçalho do pacote ICMPv6 são iguais aos do ICMP. O único ponto de atenção é o campo checksum, que é chamado, pelo Scapy de "cksum" ao invés de "chksum", como no ICMP do IPv4. Na classe implementada foi adotado o nome "checksum", para seguir o padrão com a implementação original.

4. Filtragem de pacotes

Além da funcionalidade multiplataforma e a implementação de tratamento de pacotes IPv6, também foi adicionada uma opção de filtragem de pacotes. Antes do sniffer ser iniciado, é solicitado se o usuário deseja executar algum tipo de filtro durante a execução e se deseja limitar a quantidade de pacotes a serem lidos. Opções de filtros:

- "ip": pacotes IPv4;
- "ip6": pacotes IPv6;
- "tcp": pacotes com protocolo TCP;
- "udp": pacotes com protocolo UDP;
- "icmp": pacotes com protocolo ICMP;
- "icmp6": pacotes com protocolo ICMPv6;
- "port 80": pacotes na porta 80 (pode ser usado para filtrar pacotes HTTP).

Os filtros podem ser combinados utilizando "and" ou "or". Por exemplo: "ip and port 80" ou "ip6 and tcp". Além de filtros, o usuário também tem a opção de limitar a quantidade de pacotes a serem lidos, conforme representado pela Figura 5.

```
Digite o filtro (em branco caso não deseja aplicar filtros): ip6 and udp
Digite a quantidade de pacotes que deseja ler (em branco caso não queira limite): 2

Ethernet Frame:
        - Destination: da:ac:bf:0e:6c:5e, Source: aa:73:cc:c8:4a:30, Protocol: 34525
        - IPv6 Packet:
                - Version: 6, Traffic Class: 0, Flow Label: 265216
                - Payload Length: 66, Next Header: 17, Hop Limit: 255
                - Source: fe80::41a:fdb9:3a3f:e5f0
                - Destination: ff02::fb
        - UDP Segment:
                - Source Port: 5353, Destination Port: 5353, Length: 66

Ethernet Frame:
        - Destination: da:ac:bf:0e:6c:5e, Source: aa:73:cc:c8:4a:30, Protocol: 34525
        - IPv6 Packet:
                - Version: 6, Traffic Class: 0, Flow Label: 265216
                - Payload Length: 197, Next Header: 17, Hop Limit: 255
                - Source: fe80::41a:fdb9:3a3f:e5f0
                - Destination: ff02::fb
        - UDP Segment:
                - Source Port: 5353, Destination Port: 5353, Length: 197
```

Figura 5 - filtragem e limite de pacotes

5. Resultados

Alguns exemplos de resultados de leitura de pacotes são representados pelas Figuras 6, 7 e 8. As Figuras 6 e 7 se tratam de pacotes com endereçamento IPv6, porém a 6 com protocolo TCP e a 7 com protocolo ICMPv6. Enquanto a Figura 8 mostra um pacote IPv4 de protocolo HTTP, demonstrando que, mesmo com as alterações para o Scapy, os pacotes analisados continuam mostrando os mesmos resultados.

```
Digite o filtro (em branco caso não deseja aplicar filtros): ip6 and tcp
Digite a quantidade de pacotes que deseja ler (em branco caso não queira limite): 1

Ethernet Frame:
        - Destination: da:ac:bf:0e:6c:5e, Source: aa:73:cc:c8:4a:30, Protocol: 34525
        - IPv6 Packet:
                - Version: 6, Traffic Class: 0, Flow Label: 658944
                - Payload Length: 36, Next Header: 6, Hop Limit: 64
                - Source: fe80::41a:fdb9:3a3f:e5f0
                - Destination: fe80::49c:fa07:9c76:5254
        - TCP Segment:
                - Source Port: 57661, Destination Port: 65372
                - Sequence: 3130634782, Acknowledgment: 1703184714
                - Flags:
                        - URG: 0, ACK: 1, PSH: 1
                        - RST: 0, SYN: 0, FIN:
                - TCP Data:
                        \x01\x00\x00\x00
```

Figura 6 - pacote IPv6 e TCP

```
Digite o filtro (em branco caso não deseja aplicar filtros): icmp6
Digite a quantidade de pacotes que deseja ler (em branco caso não queira limite): 1

Ethernet Frame:
        - Destination: aa:73:cc:c8:4a:30, Source: da:ac:bf:0e:6c:5e, Protocol: 34525
        - IPv6 Packet:
                - Version: 6, Traffic Class: 0, Flow Label: 0
                - Payload Length: 32, Next Header: 58, Hop Limit: 255
                - Source: fe80::49c:fa07:9c76:5254
                - Destination: fe80::41a:fdb9:3a3f:e5f0
        - ICMP Packet:
                - Type: 135, Code: 0, Checksum: 17552,
                - ICMP Data:
                        \x01\x01\xda\xac\xbf\x0e\x6c\x5e
```

Figura 7 - pacote IPv6 e ICMPv6

```
Digite o filtro (em branco caso não deseja aplicar filtros): port 80
Digite a quantidade de pacotes que deseja ler (em branco caso não queira limite): 1

Ethernet Frame:
        - Destination: 60:a4:b7:24:62:ba, Source: da:ac:bf:0e:6c:5e, Protocol: 2048
        - IPv4 Packet:
                - Version: 4, Header Length: 20, TTL: 64,
                - Protocol: 6, Source: 192.168.0.125, Target: 18.214.17.35
        - TCP Segment:
                - Source Port: 50897, Destination Port: 80
                - Sequence: 1740628765, Acknowledgment: 1606003646
                - Flags:
                        - URG: 0, ACK: 1, PSH: 1
                        - RST: 0, SYN: 0, FIN:
                - HTTP Data:
                          GET /get HTTP/1.1
                          User-Agent: PostmanRuntime/7.39.0
                          Accept: */*
                          Cache-Control: no-cache
                          Postman-Token: 82170024-a27f-4abe-80ff-42025cfc9a2b
                          Host: httpbin.org
                          Accept-Encoding: gzip, deflate, br
                          Connection: keep-alive
```

Figura 8 - pacote IPv4, TCP, na porta 80 (HTTP)