

Código Leonardo

sábado, 4 de maio de 2024 22:44



codigo_leo
nardo.py

Em relação ao formato do payload UDP da consulta, seguem orientações:

- O cliente deverá realizar apenas consultas do tipo NS (authoritative name server for the domain).
- Transaction ID (16 bits): o cliente deverá gerar um **número aleatório de 16 bits** para identificar a transação/consulta.
- Flags (16 bits): as flags de consulta deverão indicar uma consulta recursiva. Dessa forma, os 16 bits de deverão estar codificados em **0x0100**.
- Question (16 bits): deverá indicar apenas **1 consulta** e, dessa forma, deverá estar codificado em **0x0001**.
- Answer RRs, Authority RRs, Additional RRs (todos os três são campos de **16 bits**): deverão estar codificados em **0x0000**.
- Query: a query deverá informar uma consulta Type NS, Class IN para o Name informado como primeiro parâmetro da linha de comando.

codigo_leonardo.py

```
1 #!/usr/bin/env python3
2 """ Especifica o interpretador Python a ser usado para executar o script.
3     Esta linha é conhecida como "shebang" e é usada para indicar ao sistema operacional
4     qual interpretador deve ser usado para executar o script Python. Neste caso,
5     #!/usr/bin/env python3 indica que o interpretador Python 3 deve ser usado para
6     executar o script.
7 ...
8
9 import socket #biblioteca para criar socket UDP
10 import struct #biblioteca para criar pacotes UDP
11 import random
12 import sys
13 """ Importa as bibliotecas necessárias. socket é usada para criar sockets UDP, struct é
14     usada para empacotar e desempacotar dados binários, random é usada para gerar números
15     aleatórios e sys é usada para obter argumentos da linha de comando.
16     Aqui estamos importando as bibliotecas necessárias para o funcionamento do script:
17     => socket: Esta biblioteca fornece funcionalidades para criar e interagir com sockets,
18         que são usados para comunicação em rede.
19     => struct: Esta biblioteca é usada para empacotar e desempacotar dados binários.
20         É útil quando trabalhamos com protocolos de rede que enviam e recebem dados binários.
21     => random: Esta biblioteca é usada para gerar números aleatórios, o que é útil para
22         criar um ID de transação aleatório para o pacote DNS.
23     => sys: Esta biblioteca fornece acesso a algumas variáveis e funções relacionadas
24         ao sistema, como argumentos da linha de comando.
25 ...
26
27 def cria_pacote(nome):
28     """ Define uma função cria_pacote que cria um pacote DNS com o nome de domínio
29         especificado. Ele gera um ID de transação aleatório e define várias flags.
30         Em seguida, adiciona o nome de domínio ao pacote.
31         Esta função cria_pacote é responsável por criar um pacote DNS com base no nome
32         de domínio fornecido como argumento.
33     ...
34     # Cabeçalho: [Transaction ID, Flags, QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT]
35     # Transaction ID eh um numero aleatorio
36     # Flag 0x0100 para busca DNS padrao
37     # QDCOUNT = 1
38     busca = struct.pack(">HHHHHH", random.getrandbits(16), 0x0100, 0x0001, 0x0000, 0x0000,
39                         0x0000)
40     """ => struct.pack("HHHHHH", ...): Empacota os campos do cabeçalho do pacote DNS
41         de acordo com o formato especificado. Os campos são: ID de transação, Flags,
42         QDCOUNT (número de perguntas), ANCOUNT (número de respostas), NSCOUNT (número
43         de registros de autoridade) e ARCOUNT (número de registros adicionais).
44     ...
45     """ => random.getrandbits(16): Gera um número aleatório de 16 bits, que será usado
46         como ID de transação no cabeçalho do pacote DNS.
47     ...
48
49     # Colocando o domínio no padrão de busca DNS
50     dominios = nome.split(".")
51     """ => nome.split("."): Divide o nome de domínio em uma lista de subdomínios.
52     ...
53     for dominio in dominios:
54         busca += struct.pack("B", len(dominio)) + dominio.encode()
55         busca += struct.pack("B", 0x0000) # Fim do hostname
56     """ => struct.pack("B", len(dominio)) + dominio.encode(): Empacota cada subdomínio
57         em formato de comprimento de subdomínio seguido pelos próprios subdomínios.
58         Os subdomínios são codificados em bytes antes de serem empacotados.
59     ...
```

* FORMATO: >HHHHHH = FORMATAÇÃO DA STRING. ELA ESPECIFICA COMO OS DADOS DEVEM SER EMPACOTADOS.

→ >: EMPACOTAMENTO EM ORDEM BIG-ENDIAN, OS BYTES + SIGNIFICATIVOS VEM PRIMEIRO
→ H: UNSIGNED SHORT INTEGER (2BYTES), SÃO 6 VALORES DE 16 BITS.

* FLAG: 0x0100 = SOLICITAÇÃO DE BUSCA DNS PADRÃO.

→ O MSB é 1 (0x01) INDICANDO QUE É UMA MENSAGEM DE SOLICITAÇÃO (QUERY)
→ O LSB é 0 (0x00) INDICANDO QUE A MSG É UM PACOTE NORMAL (NÃO DE RESPOSTA)

* QDCOUNT: 0x0001 = REPRESENTA O NÚMERO DE PREGUNTAS INCLUIDAS NO PACOTE DNS.
→ É IGUAL A 1: APENAS UMA PREGUNTA NO PACOTE, QUE É PARA O NOME DO DOMÍNIO ESPECIFICADO.

* ANCOUNT : } NSCOUNT : } 0x0000 = REPRESENTAM ARCOUNT : }

Nº DE RESPOSTAS
REGISTROS DE AUTORIDADE E REGISTROS ADICIONAIS NO PROTO DNS.

→ TODOS SÓ O PORQUE O PACOTE É UMA SOLICITAÇÃO E NÃO UMA RESPOSTA.

* FORMATO: B → representa um byte.
→ empacota o comprimento len(dominio) como um byte e adiciona ao pacote "BUSCA".

* dominio.encode(): converte o nome do domínio para uma sequência de bytes usando UTF-8 (codificação) e adiciona ao pacote.

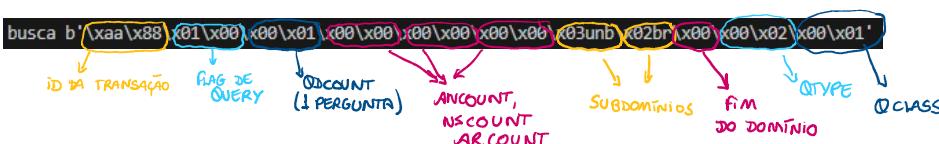
* dommine_encode(): converte o nome do domínio para uma sequência de bytes usando UTF-8 (padrão) e adiciona ao pacote.

* "0", 0x0000: é para indicar o final do nome do domínio no pacote DNS.

```
=====
inicio main - obtém argumentos
dominio unb.br
dns_server 8.8.8.8

=====
Cria um pacote DNS
dominios ['unb', 'br']

busca b'\xaa\x88\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00'
busca b'\xaa\x88\x01\x00\x00\x01\x00\x00\x00\x00\x00\x03unb'
busca b'\xaa\x88\x01\x00\x00\x01\x00\x00\x00\x00\x00\x03unb\x02br'
busca b'\xaa\x88\x01\x00\x00\x01\x00\x00\x00\x00\x00\x03unb\x02br\x00'
busca b'\xaa\x88\x01\x00\x00\x01\x00\x00\x00\x00\x00\x03unb\x02br\x00\x02\x00\x01'
```



A implementação do cliente deverá utilizar o protocolo UDP como protocolo de transporte.

- Como o protocolo UDP é orientado a mensagem, o cliente deverá aguardar 2 segundos pela chegada da resposta. Caso a resposta não tenha chegado, uma nova tentativa deverá ser realizada.
- O cliente tentará resolver o nome por até 3 vezes. Depois da terceira vez, uma mensagem de erro deverá ser informada ao usuário.
- Caso o nome apresentado pelo usuário não exista, o usuário deverá ser informado mediante mensagem em console.
- O primeiro argumento da linha de comando deverá informar o nome cuja resolução se procura. O segundo argumento informará o IP do servidor DNS que será consultado.
 - O servidor funcionará na porta padrão para o serviço DNS: UDP 53

```
60     # QTYPE=2 para consulta NS, QCLASS=1 para IN (Internet)
61     busca += struct.pack("HH", 0x0002, 0x0001)
62     """ > struct.pack(">HH", 0x0002, 0x0001): Empacota os campos QTYPE (tipo de
63     pergunta) e QCLASS (classe de pergunta) para especificar que estamos fazendo
64     uma consulta NS (Name Server) na classe IN (Internet).
65
66     return busca
67
68 def envia_e_recebe(busca, dns_server):
69     """ Define uma função envia_e_recebe que envia a busca DNS para o servidor
70     especificado e recebe a resposta. Ele tenta enviar e receber a busca até
71     três vezes, com um intervalo de tempo limite de 2 segundos.
72     Esta função envia_e_recebe é responsável por enviar a busca DNS para o
73     servidor DNS especificado e receber a resposta.
74
75     for i in range(3):
76         """ O loop for tenta enviar e receber a busca até três vezes em caso de falha.
77         """ → CRIA UM SOCKET → ESPECIFICA O ESCOPO DE INTERNET: FAMÍLIA DE ENDEREÇOS IPv4.
78         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) → ESTILO DE COMUNICAÇÃO DO TIPO DATAGRAMA (USADO PARA COMUNICAÇÃO DE
79         """ > socket.socket(socket.AF_INET, socket.SOCK_DGRAM): Cria um socket UDP
80         para comunicação com o servidor DNS.
81
82         sock.settimeout(2) # timeout = 2s → DEFINE O TEMPO LIMITE DE 2SEGUNDOS DE ESPERA PARA OPERAÇÕES DE
83         """ > sock.settimeout(2): Define um tempo limite de 2 segundos para esperar
84         pela resposta do servidor DNS.
85         """ → ENVIA DADOS PARA O DESTINO ESPECIFICADO. ← dados: Busca
86         sock.sendto(busca, (dns_server, 53)) #envia busca DNS para o servidor DNS, porta 53
87         """ > sock.sendto(busca, (dns_server, 53)): Envia a busca DNS para o servidor
88         DNS na porta 53.
89         ... função que tenta receber os dados do socket, até 512 bytes.
90         try:
91             resposta = sock.recvfrom(512)[0] # receber resposta do servidor DNS (tamanho
92             """ maximo de 512 bytes)
93             """ > sock.recvfrom(512)[0]: Recebe a resposta do servidor DNS.
94             A resposta é limitada a 512 bytes.
95
96             return resposta → resposta é uma tupla: dados recebidos+endereço do
97             except socket.timeout: remetente
98                 continue → Se o tempo for > 2s, continua e tenta de novo
99
100            """
```

* TIPO DE CONSULTA NS (TYPE NS)

→ 0x0002 = QTYPE

* CLASSE DE CONSULTA IN (CLASS IN)

→ 0x0001 = QCLASS

```

94     ...
95     return resposta → resposta é uma tupla: dados recebidos + endereço do
96     except socket.timeout: → vermelho
97         continue → base o tempo for > 2 seg continua e tenta de novo
98
99     return None #caso não haja resposta, retorna vazio para o parser
100
101 def parse_resposta(resposta, hostname):
102     """ Define uma função parse_resposta que analisa a resposta DNS recebida (ou seja,
103     Verifica se a resposta é nula: imprime uma mensagem de erro se for completo)
104     Esta função parse_resposta é responsável por analisar a resposta DNS recebida
105     e extrair informações relevantes dela.
106
107     if resposta is None:
108         ... Se a resposta for nula, imprime uma mensagem de erro indicando que não
109         foi possível coletar a entrada NS para o domínio especificado.
110
111     print("Nao foi possivel coletar entrada NS para", hostname)
112     return
113
114 # Parsing do cabeçalho
115 cabecalho = struct.unpack('>HHHHH', resposta[:12]) → desempacota obe de cendo esse formato.
116     """ Desempacota o cabeçalho da resposta DNS, que contém várias informações,
117     como ID de transação, flags e contadores.
118     Esta linha desempacota os primeiros 12 bytes da resposta DNS para extrair
119     o cabeçalho, que contém informações como ID de transação, flags e contadores.
120
121 #print("Transaction ID:", cabecalho[0])

```

indicativo de falha na comunicação com o servidor DNS.

```

===== Envia o pacote DNS =====
pacote enviado
b'\xa2B\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\r\x17\x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\r\x17\x00
\x07\x04dns2\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\r\x17\x00\x07\x04dns3\xc0\x0c'

===== Analisa a resposta DNS recebida =====
12 primeiros bytes da resposta DNS
0. Transaction ID: 12168
1. Flags: 0b10000011000000
2. Questions: 1
3. Answers RRs: 3
4. Authority RRs: 0
5. Additional RRs: 0

```

```

===== Analisa a resposta DNS recebida =====
12 primeiros bytes da resposta DNS
0. Transaction ID: 12168
1. Flags: 0b10000011000000
2. Questions: 1
3. Answers RRs: 3
4. Authority RRs: 0
5. Additional RRs: 0

```

```

122     #print("Flags:", bin(cabecalho[1])) *
123     # RCODE - últimos 4 bits: (0 - sem erro) (1 - format error) (2 - serv. error) (3 - ..
124     #name error) (4 - query não implementado) (5 - recusado) *
125     #print("Questions:", cabecalho[2])
126     #print("Answers:", cabecalho[3])
127     #print("Authority RRs:", cabecalho[4])
128     #print("Additional RRs:", cabecalho[5])
129
130     #Flag não aponta erro, mas não há respostas NS
131     if ((0b0000000000001111 & int(cabecalho[1])) == 0b0000000000000000) and (cabecalho[3] == 0): *
132         ... Verifica se o cabeçalho indica que não há erro e se o contador de respostas
133         é zero. Se ambos forem verdadeiros, imprime uma mensagem indicando que o
134         domínio não possui entrada NS.
135         Esta parte do código verifica se o cabeçalho da resposta indica que não
136         há erros (0b0000000000001111 & int(cabecalho[1]) == 0b0000000000000000) e se o
137         contador de respostas é zero. Se ambos os critérios forem verdadeiros,
138         imprime uma mensagem indicando que o domínio não possui entrada NS.
139
140     print("Dominio", hostname, "nao possui entrada NS") → não houve erro e resposta veio zerada.
141     return
142
143     #Flag aponta erro de nome → máscara p/ últimos 4 bits
144     elif (0b0000000000001111 & int(cabecalho[1]) == 0b0000000000000011): *
145         ... Este bloco de código verifica se o bit RCODE (Return Code) do cabeçalho DNS
146         indica um erro de nome (0b0000000000001111). Se isso for verdade, significa
147         que o domínio especificado não foi encontrado. Nesse caso, uma mensagem de

```

maioria de bits
verifica-se se os últimos 4 bits
do cabeçalho [1] é igual
a zero → ou seja: o flag tem os
últimos 4 bits = 0 - Não erro*
ANCOUNT = 0 → qTotal
de respostas = 0
FLAG = 3 - NAME ERROR

O RCODE (Return Code) é um campo de 4 bits no cabeçalho de uma resposta DNS que indica o

```

142 #Flag aponta erro de nome ...
143 elif(0b00000000000000111 & int(cabecalho[1]) == 0b0000000000000011):
144     *** Este bloco de código verifica se o bit RCODE (Return Code) do cabeçalho DNS
145     indica um erro de nome (0b0000000000000011). Se isso for verdade, significa
146     que o domínio especificado não foi encontrado. Nesse caso, uma mensagem de
147     erro é impressa indicando que o domínio não foi encontrado.
148 ...
149     print("Dominio", hostname, "nao encontrado")
150     return
151
152 #Flag aponta erro de servidor
153 elif(0b00000000000000111 & int(cabecalho[1]) == 0b0000000000000010):
154     *** Este bloco de código verifica se o bit RCODE do cabeçalho DNS indica um erro
155     de servidor (0b0000000000000010). Se isso for verdade, significa que ocorreu
156     um erro no servidor DNS ao coletar a entrada NS. Nesse caso, uma mensagem de
157     erro é impressa indicando que não foi possível coletar a entrada NS para o
158     domínio especificado.
159 ...
160     print("Nao foi possivel coletar entrada NS para", hostname)
161     return
162
163 offset = 12 # offset apos o cabecalho
164 *** Define o offset inicial apos o cabeçalho DNS que é de 12 bytes.
165 ...
166
167 # Pular o campo de Questions
168 *** Este bloco de código é responsável por pular o campo de perguntas (Questions)
169 #resposta DNS! Ele avança o offset até encontrar um byte nulo, indicando o
170 final do campo de perguntas, e então avança mais 5 bytes para pular o restante
171 do campo.
172 ...
173 while resposta[offset] != 0:
174     offset += 1 # Pula o QName
175     offset += 5 # pula o null apos o fim do nome
176
177 # Parser do campo de Anwers
178 *** Este loop itera sobre o número de respostas (Answers) no cabeçalho DNS. ***
179 for i in range(cabecalho[3]): # cabecalho[3] indica a quantidade de respostas
180     if resposta[offset] >= 192: # 192 = 0xC0 verifica inicio de um ponteiro
181         *** Este bloco de código lida com os ponteiros na resposta DNS. Se o byte
182         atual for maior ou igual a 192 (indicando um ponteiro), avanca-se 2 bytes
183         para pular o ponteiro. Caso contrário, avanca-se ate encontrar um byte

```

FLAG = 3 - NAME ERROR

FLAG = 4 → QUERY NÃO IMPLEMENTADA

*TAMANHO VARIÁVEL ***

PULA O CABEÇALHO E OS SUBDOMÍNIOS

e mais 4 bytes. (QTYPE e QCLASS)

O RCODE (Return Code) é um campo de 4 bits no cabeçalho de uma resposta DNS que indica o resultado da consulta. Aqui estão os possíveis valores de RCODE e seus significados:

0. (0000): Sem erro - Indica que a operação foi concluída com sucesso.
1. (0001): Erro de formato - Indica que a consulta DNS tem um formato inválido.
2. (0010): Erro de servidor - Indica que ocorreu um erro no servidor DNS.
3. (0011): Erro de nome - Indica que o nome de domínio especificado não foi encontrado.
4. (0100): Consulta não implementada - Indica que o tipo de consulta especificado não é suportado pelo servidor DNS.
5. (0101): Recusado - Indica que o servidor DNS recusou a consulta por algum motivo.

Esses códigos são usados para fornecer informações sobre o status da consulta DNS e ajudar a diagnosticar problemas de comunicação ou configuração.

O resultado da consulta DNS bem sucedida deverá ser informado na saída padrão (stdout do console) e deverá ter o seguinte formato:

- nome_domínio ◊ nome_servidor_email

Para informar o resultado da consulta, será necessário interpretar a mensagem retornada pelo servidor e extraí a informação do nome do servidor de e-mail de seu payload.

- Lembre-se que, entradas NS costumam retornar várias respostas autoritativas.

pão de
TRANSMIÇÃO
FLAG

resposta recebida:

```

b'(\xc7\x91\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02
\x00\x01\x0c\x00\x02\x00\x01\x00\x00\x01\xe3\x00\x07\x04dns2\xc0\x0c\x00
\x0c\x02\x00\x01\x00\x00\x01\x00\x01\xe3\x00\x07\x04dns3\xc0\x0c\x00\x00\x02
\x00\x01\x00\x00\x01\xe3\x00\x07\x04dns1\xc0\x0c'
=====
```

offset +5 → 8 bits
NULL → 16 bits
QTYPE → 16 bits
QCLASS → 16 bits

PONTEIRO

TYPE

ANSWER, NS ANSWER, ARCOUNT

CLASS e TTL

RDLENGTH

*tamanho variável ***

```

184     pula(indicando o final do nome) e mais um byte para pular o byte nulo
185     ...
186     offset += 2 # Pula os 2 bytes de ponteiro
187 else:
188     while resposta[offset] != 0:
189         offset += 1 # Pula o nome
190     offset += 1 # Pula o null
191
192     tipo = struct.unpack(">H", resposta[offset:offset+2])[0] #extrai o campo TYPE
193     ... Extrai o campo de tipo (TYPE) da resposta DNS. ...
194
195     offset += 8 #pula campos TYPE, CLASS e TTL
196     tamanho= struct.unpack(">H", resposta[offset:offset+2])[0] #extrai o campo RDLENGTH
197     offset += 2 #pula campo RDLENGTH
198     ... Avança o offset para pular os campos TYPE, CLASS e TTL, e então extrai o ...
199     ... campo de comprimento de dados (RDLENGTH) da resposta DNS.
200     ...
201
202     #offset += 10 #move o offset para depois do campo de tamanho
203     if tipo == 2: # tipo = 2 eh NS
204         ... Se o tipo for 2 (indicando que é uma resposta NS), chama a função parse_domain ...
205         ... para analisar o nome do servidor.
206
207     dominios = []
208     nome_srv = parse_domain(resposta, offset, dominios)
209     print(hostname, "<", nome_srv)
210
211     offset += tamanho
212     ... Avança o offset para o próximo registro na resposta DNS, com base no ...
213     ... comprimento do registro atual.
214
215
216 def parse_domain(resposta, offset, dominios):
217     """ Define uma função parse_domain que analisa o campo de nome de domínio na ...
218     ... resposta DNS. Ele usa um loop para percorrer o nome do domínio e lidar com ...
219     ... ponteiros se necessário. """
220     Esta função parse_domain é definida para analisar o campo de nome de domínio
221     na resposta DNS. Ela recebe três argumentos: resposta (a resposta DNS que será
222     analisada), offset (o deslocamento a partir do qual a análise será iniciada) e
223     dominios (uma lista para armazenar os componentes do nome de domínio).
224     ...
225     while True:
226         ... Inicia um loop infinito para percorrer os componentes do nome de domínio ...
227         ... na resposta DNS
228         ...
229         tam = resposta[offset]
230         ... Obtém o tamanho do próximo componente do nome de domínio. ...
231         if tam == 0:
232             ... Se o tamanho for 0, indica o final do nome de domínio. Portanto,
233             ... incrementa o offset em 1 e sai do loop.
234             ...
235         offset += 1
236         break
237     elif tam >= 192: # eh ponteiro
238         ... Se o tamanho for maior ou igual a 192, indica um ponteiro para um nome ...
239         ... de domínio anteriormente encontrado. Então, desempacota os próximos ...
240         ... bytes como um inteiro sem sinal (unsigned short) no formato big-endian
241         ... e ajusta o ponteiro com a máscara 0xFFFF. Em seguida, chama
242         ... recursivamente a função parse_domain com o novo ponteiro como offset.
243         ...
244         ponteiro = struct.unpack(">H", resposta[offset:offset+2])[0]
245         offset += 2

```

```

246     ponteiro = ponteiro & 0x3FFF #Ajustar ponteiro
247     return parse_domain(resposta, ponteiro, dominios) #passa o ponteiro como offset
para ler o domínio
248     else:
249         ''' Se o tamanho não for 0 nem indicar um ponteiro, então é um componente
250             do nome de domínio. Incrementa o offset em 1, adiciona o componente do
251             nome de domínio à lista dominios, após decodificá-lo de bytes para string
252             e, em seguida, incrementa o offset pelo tamanho do componente.'''
253         ...
254         offset += 1
255         dominios.append(resposta[offset:offset+tam].decode()) #coloca o domain no nome
256         offset += tam
257     ''' Após sair do loop, junta todos os componentes do nome de domínio na lista
258         dominios, separados por ponto, e retorna o nome de domínio completo.'''
259     ...
260
261     return '.'.join(dominios)
262
263
264 args = sys.argv[1:]
265
266 dominio = args[0]
267 dns_server = args[1]
268 ''' Obtém os argumentos da linha de comando: o nome do domínio e o servidor DNS.'''
269
270 pacote = cria_pacote(dominio)
271 ''' Cria um pacote DNS para a consulta ao nome de domínio especificado.'''
272
273 resposta = envia_e_recebe(pacote, dns_server)
274 ''' Envia o pacote DNS para o servidor DNS especificado e recebe a resposta.'''
275
276 parse_resposta(resposta, dominio)
277 ''' Analisa a resposta DNS recebida usando a função parse_resposta, passando a
278     resposta e o nome de domínio como argumentos.'''
279
280

```

```
=====
inicio main - obtém argumentos
-----
domínio: unb.br
dns_server: 8.8.8.8
=====

=====
Cria um pacote DNS
-----
domínios ['unb', 'br']

busca b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00'
busca b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03unb'
busca b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03unb\x02br'
busca b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x03unb\x02br\x00'
busca b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x03unb\x02br\x00\x02\x00\x01'
=====

=====
Envia o pacote DNS
-----
pacote enviado
b'P\x1f\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x03unb\x02br\x00\x02\x00\x01'

resposta recebida:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x00\x02\x00\x01\x00\x00\x00\xce\x00
\x07\x04dns1\xc0\x0c\xc0\x00\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'
```

```

=====
Analisa a resposta DNS recebida
-----
cabecalho: b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00'

12 primeiros bytes da resposta DNS
-----
0. Transaction ID: 20511
1. Flags: 0b1000000110000000
2. Questions: 1
3. Answers RRs: 3
4. Authority RRs: 0
5. Additional RRs: 0

-----
offset= 12 ( 6 - 16bits ) - cabecalho:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00'

-----
offset= 24 ( 12 - 16bits ) - cabecalho + subdominios + null + QTYPE + QCLASS:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01'

-----
offset= 26 ( 13 - 16bits - encontrou o ponteiro x0c):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c'

-----
offset= 34 ( 17 - 16bits ) - 2 ultimos bytes eh o tipo:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce'

tipo: 2

-----
offset= 36 ( 18 - 16bits ) - 2 ultimos bytes eh o tamanho:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07'

tamanho: 7

-----
unb.br <> dns3.unb.br

-----
offset= 43 ( 21 - 16bits - incluindo o tamanho par recomendar):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c'

-----
resposta:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\x00\x00\x00\x00\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'
=====
```

```

offset= 45 ( 22 - 16bits - encontrou o ponteiro x0c):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c'

offset= 53 ( 26 - 16bits) - 2 ultimos bytes eh o tipo:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce'
tipo: 2

offset= 55 ( 27 - 16bits) - 2 ultimos bytes eh o tamanho:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07'
tamanho: 7

unb.br <> dns1.unb.br

offset= 62 ( 31 - 16bits - incluindo o tamanho par recomendar):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c'

=====

resposta:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'

=====

```

```

offset= 64 ( 32 - 16bits - encontrou o ponteiro x0c):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\x0c'

offset= 72 ( 36 - 16bits) - 2 ultimos bytes eh o tipo:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\x00\x07'

tipo: 2

offset= 74 ( 37 - 16bits) - 2 ultimos bytes eh o tamanho:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'

tamanho: 7

unb.br <> dns2.unb.br

offset= 81 ( 40 - 16bits - incluindo o tamanho par recomendar):
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'

=====

resposta:
b'P\x1f\x81\x80\x00\x01\x00\x03\x00\x00\x00\x03unb\x02br\x00\x00\x02\x00\x01\xc0\x0c\x00
\x02\x00\x01\x00\x00\x00\xce\x00\x07\x04dns3\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\
x00\x07\x04dns1\xc0\x0c\xc0\x0c\x00\x02\x00\x01\x00\x00\xce\x00\x07\x04dns2\xc0\x0c'

=====
=====
```