BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Network Applications and Network Administration

# Tunneling of data transmissions via DNS queries

**Ivan Tsiareshkin, xtsiar00**

November 14, 2022

## Contents

## 1 Abstract

DNS-tunneling is a method of cyber attack that encodes the data of other programs or protocols in DNS queries and responses. DNS tunneling often includes data payloads that can be added to an attacked DNS server and used to control a remote server and applications.
The essence of data transmission through DNS queries is that the data can be encoded in the query name field, which consists of several labels separated by a dot. The length of the entire domain name must not exceed 255 bytes, the length of each label must not exceed 63 bytes. Only alphanumeric characters and hyphens are allowed. Specifically in this project, the goal is to create client and server aplication, that will communicate with each other via UDP. Using that protocol, the client will send encoded data to the server. The data will be encoded so, that they will represent a valid DNS query. Moreover, the end of the DNS-query will contain the {BASE_HOST}, it will not be encoded and will serve as a flag for the server that this packet is for him and he needs to process it. he server will receive the necessary DNS-queries and process them, saving the transmitted data to the specified file.

## 2 Implementation

The project is written in C language using network libraries. UDP protocol is used for data transfer and base32 for encoding/decoding.

### 2.1 Client - dns_sender.c

#### 2.1.1 Arguments

The client program will be able to be started and controlled using the following parameters: `dns_sender [-u UPSTREAM_DNS_IP] {BASE_HOST} {DST_FILEPATH} [SRC_FILEPATH]`

- [UPSTREAM_DNS_IP] is optional. If argument is provided, sender will validate it by the inet_aton function. If not provided, sender will send data to the address of internal DNS resolver of `systemd-resolved` provided by `resolv.conf`.
- {BASE_HOST} is mandatory. It should consist of two labels separated by a ".". Each label's length () should not exceed 63 characters and `base host` should not start or end with "." character.
- {DST_FILEPATH} is mandatory.
- [SRC_FILEPATH] is optional. If argument is provided, sender checks if file exists. If not provided, sender will read data from standard input (stdin).

#### 2.1.2 Return codes and errors

- 0 - Program completed successfully
- 1 - File read or write error
- 2 - Wrong parameter
- 3 - socket() function error

- 4 - sendto() function error
- 5 - recvfrom() function error

### 2.1.3 Main function - main

The main function of the client firstly reads and processes the incoming arguments, checking some of them for validity. After that, the program sends the first DNS-query to the server, which will always contains only the DST_FILEPATH.
After the first DNS-query with filepath is successfully sent, the main loop of the program begins. In this loop, data is gradually read for transmission to the server. If the incoming file contains more dates than the one DNS-query can contain (limit of 100 bytes of the raw date), then the incoming file is split into several DNS-queries and they will be gradually sent to the server

### 2.1.4 Send function - send_chunk

The send_chunk function creates a valid DNS-query and sends it to the server.
The function sets the required values of the variables in the header of the DNS-query, then splits the piece of encoded data in base32 into labels, the length of which can be maximum 63, then adds the base host.
The function sends a ready DNS-query via the UDP protocol and waits for a response from the server if the packet was received by it.

## 2.2 Server - dns_receiver

The client program will be able to be started and controlled using the following parameters:

```
dns_receiver {BASE_HOST} {DST_DIRPATH}
```

- {BASE_HOST} is mandatory. It should consist of two labels separated by a ”.”. Each label's length () should not exceed 63 characters and `base host` should not start or end with ”.” character.
- {DST_DIRPATH} is mandatory.

### 2.2.1 Return codes and errors

- 0 - Program completed successfully
- 2 - Wrong parameter
- 3 - socket() function error
- 4 - sendto() function error
- 5 - recvfrom() function error
- 6 - bind() function error
- 7 - mkdir() function error

### 2.2.2 Main function - main

The main function of the server firstly reads and processes the incoming arguments, checking some of them for validity. After that, it opens the socket and waits for incoming packets in the main loop.
After receiving the packet, the server extracts the encoded data along with the BASE_HOST from the DNS-query and checks whether the received BASE_HOST is identical to the one that was passed as an argument.
If yes, then the server starts processing the data. Looking at the DNS-query id (header-¿id), the server understands whether the package is with the DST_FILEPATH or whether it is a package with data that it needs to save. If it's a packet with DST_FILEPATH, then the server decodes the data, extracts the filepath, checks its format, and merges it with the DST_DIRPATH. The final path to the file where the data will be saved will be {DST_DIRPATH}/{DST_FILEPATH}. After the server will send a response to the client that the data was received successfully.
In case of accepting a packet with data that needs to be saved, the server will call the write_data function, which will extract the data and write it to the necessary file.

### 2.2.3 Write function - write_data

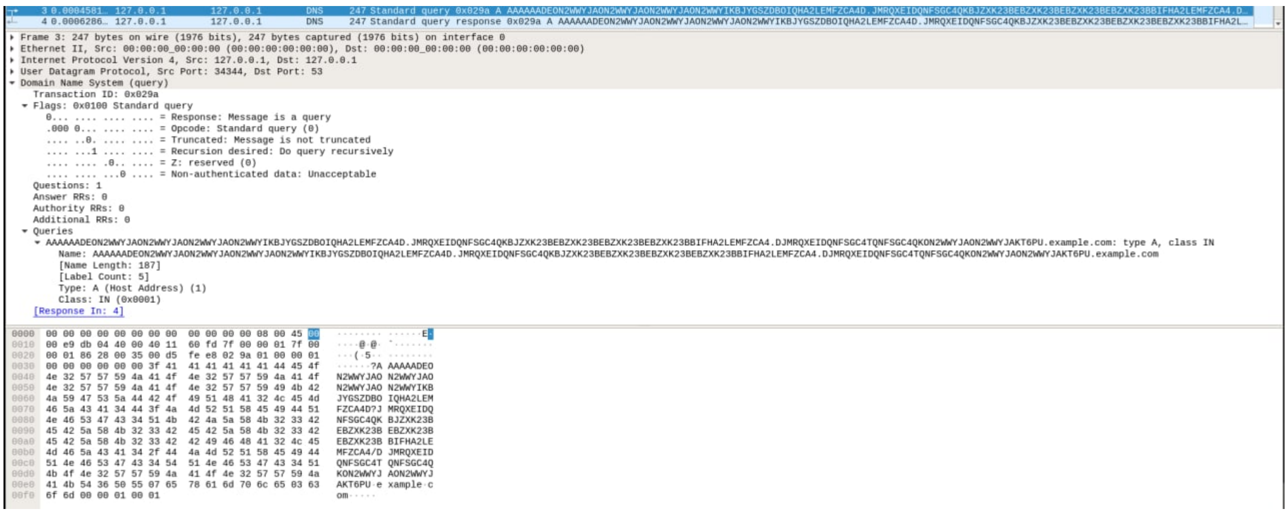
This function decodes the incoming packet and using the fwrite function, which is open in a + b mode, will write the data to the end of the file.

## 2.3 Encoding/Decoding - base32.h/.c

A solution from Google is used to encode and decode data. This decision only uses capitalised latin characters plus numbers from 2 to 7.

## 3 Testing and display functionality

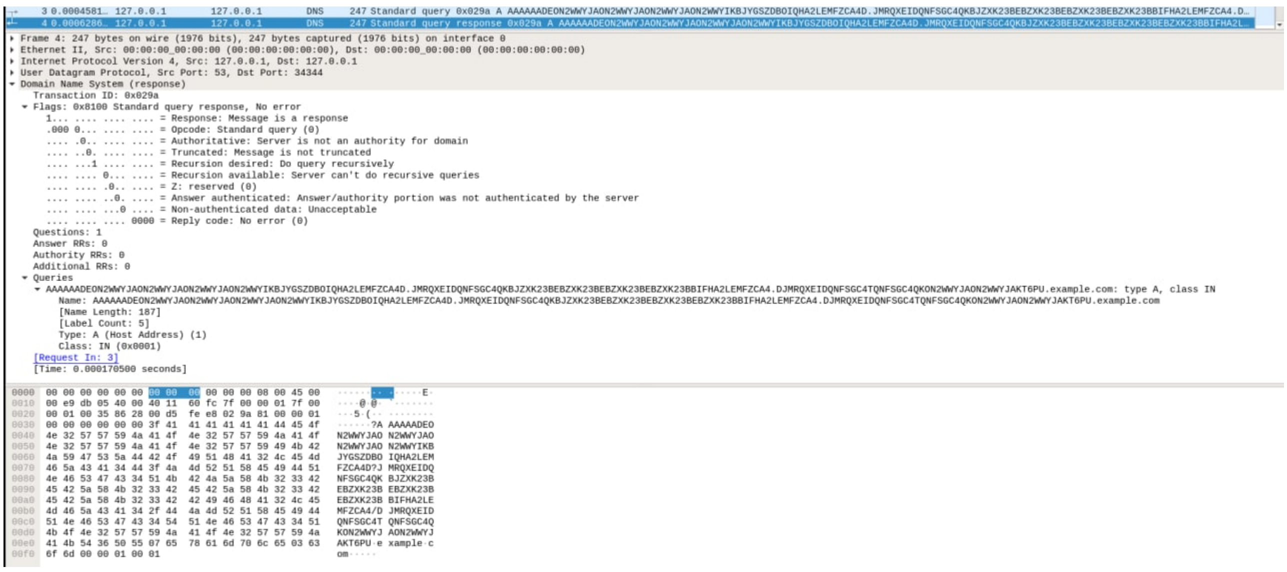
To test the general work of the client and server, I sent various files of various formats and sizes. Edge cases of programs were also tested.
This is how the transmission of one DNS-query to the server and its DNS-response to the client looks like:



```
  3 0.0004581… 127.0.0.1    127.0.0.1    DNS   247 Standard query 0x029a A AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.D…
  4 0.0006286… 127.0.0.1    127.0.0.1    DNS   247 Standard query response 0x029a A AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2L…
▸ Frame 3: 247 bytes on wire (1976 bits), 247 bytes captured (1976 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 34344, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x029a
  ▾ Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.DJMRQXEIDQNFSGC4TQNFSGC4QKON2WWYJAON2WWYJAKT6PU.example.com: type A, class IN
        Name: AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.DJMRQXEIDQNFSGC4TQNFSGC4QKON2WWYJAON2WWYJAKT6PU.example.com
        [Name Length: 187]
        [Label Count: 5]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Response In: 4]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 e9 db 04 40 00 40 11  60 fd 7f 00 00 01 7f 00   ····@·@· `·······
0020  00 01 86 28 00 35 00 d5  fe e8 02 9a 01 00 00 01   ···(·5·· ········
0030  00 00 00 00 00 00 3f 41  41 41 41 41 41 44 45 4f   ······?A AAAAADEO
0040  4e 32 57 57 59 4a 41 4f  4e 32 57 57 59 4a 41 4f   N2WWYJAO N2WWYJAO
0050  4e 32 57 57 59 4a 41 4f  4e 32 57 57 59 49 4b 42   N2WWYJAO N2WWYIKB
0060  4a 59 47 53 5a 44 42 4f  49 51 48 41 32 4c 45 4d   JYGSZDBO IQHA2LEM
0070  46 5a 43 41 34 44 3f 4a  4d 52 51 58 45 49 44 51   FZCA4D?J MRQXEIDQ
0080  4e 46 53 47 43 34 51 4b  42 4a 5a 58 4b 32 33 42   NFSGC4QK BJZXK23B
0090  45 42 5a 58 4b 32 33 42  45 42 5a 58 4b 32 33 42   EBZXK23B EBZXK23B
00a0  45 42 5a 58 4b 32 33 42  42 49 46 48 41 32 4c 45   EBZXK23B BIFHA2LE
00b0  4d 46 5a 43 41 34 2f 44  4a 4d 52 51 58 45 49 44   MFZCA4/D JMRQXEID
00c0  51 4e 46 53 47 43 34 54  51 4e 46 53 47 43 34 51   QNFSGC4T QNFSGC4Q
00d0  4b 4f 4e 32 57 57 59 4a  41 4f 4e 32 57 57 59 4a   KON2WWYJ AON2WWYJ
00e0  41 4b 54 36 50 55 07 65  78 61 6d 70 6c 65 03 63   AKT6PU·e xample·c
00f0  6f 6d 00 00 01 00 01                               om·····
```

Figure 1: DNS-query



```
  3 0.0004581… 127.0.0.1    127.0.0.1    DNS   247 Standard query 0x029a A AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.D…
  4 0.0006286… 127.0.0.1    127.0.0.1    DNS   247 Standard query response 0x029a A AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2L…
▸ Frame 4: 247 bytes on wire (1976 bits), 247 bytes captured (1976 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 53, Dst Port: 34344
▾ Domain Name System (response)
    Transaction ID: 0x029a
  ▾ Flags: 0x8100 Standard query response, No error
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .0.. .... .... = Authoritative: Server is not an authority for domain
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... 0... .... = Recursion available: Server can't do recursive queries
      .... .... .0.. .... = Z: reserved (0)
      .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
      .... .... ...0 .... = Non-authenticated data: Unacceptable
      .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.DJMRQXEIDQNFSGC4TQNFSGC4QKON2WWYJAON2WWYJAKT6PU.example.com: type A, class IN
        Name: AAAAAADEON2WWYJAON2WWYJAON2WWYJAON2WWYIKBJYGSZDBOIQHA2LEMFZCA4D.JMRQXEIDQNFSGC4QKBJZXK23BEBZXK23BEBZXK23BEBZXK23BBIFHA2LEMFZCA4.DJMRQXEIDQNFSGC4TQNFSGC4QKON2WWYJAON2WWYJAKT6PU.example.com
        [Name Length: 187]
        [Label Count: 5]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    [Request In: 3]
    [Time: 0.000170500 seconds]

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 e9 db 05 40 00 40 11  60 fc 7f 00 00 01 7f 00   ····@·@· `·······
0020  00 01 00 35 86 28 00 d5  fe e8 02 9a 81 00 00 01   ···5·(·· ········
0030  00 00 00 00 00 00 3f 41  41 41 41 41 41 44 45 4f   ······?A AAAAADEO
0040  4e 32 57 57 59 4a 41 4f  4e 32 57 57 59 4a 41 4f   N2WWYJAO N2WWYJAO
0050  4e 32 57 57 59 4a 41 4f  4e 32 57 57 59 49 4b 42   N2WWYJAO N2WWYIKB
0060  4a 59 47 53 5a 44 42 4f  49 51 48 41 32 4c 45 4d   JYGSZDBO IQHA2LEM
0070  46 5a 43 41 34 44 3f 4a  4d 52 51 58 45 49 44 51   FZCA4D?J MRQXEIDQ
0080  4e 46 53 47 43 34 51 4b  42 4a 5a 58 4b 32 33 42   NFSGC4QK BJZXK23B
0090  45 42 5a 58 4b 32 33 42  45 42 5a 58 4b 32 33 42   EBZXK23B EBZXK23B
00a0  45 42 5a 58 4b 32 33 42  42 49 46 48 41 32 4c 45   EBZXK23B BIFHA2LE
00b0  4d 46 5a 43 41 34 2f 44  4a 4d 52 51 58 45 49 44   MFZCA4/D JMRQXEID
00c0  51 4e 46 53 47 43 34 54  51 4e 46 53 47 43 34 51   QNFSGC4T QNFSGC4Q
00d0  4b 4f 4e 32 57 57 59 4a  41 4f 4e 32 57 57 59 4a   KON2WWYJ AON2WWYJ
00e0  41 4b 54 36 50 55 07 65  78 61 6d 70 6c 65 03 63   AKT6PU·e xample·c
00f0  6f 6d 00 00 01 00 01                               om·····
```

Figure 2: DNS-response

## 3.1 Used sources

1. https://mislove.org/teaching/cs4700/spring11/handouts/project1-primer.pdf
2. https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/UDPSockets.html
3. https://www.ietf.org/rfc/rfc1035.txt
4. https://datatracker.ietf.org/doc/html/rfc1034
5. https://github.com/google/google-authenticator-libpam/tree/master/src