DNS Monitor

ISA Project 2024/2025

Author: Denis Fekete

Table of Contents

1. About

DNS-Monitor is a command line program for storing DNS communication from packets and displaying them in human readable form. Program uses system network interfaces for live capturing of packets or can open already stored communication from a file. Program is also capable of storing captured communication into files, more specifically it can store domain names, that were present in communication, and also translations from IP addresses to domain names. Program is written in C using GNU17 standard version with PCAP library.

2. Theory

Communication through internet today is done by stacking multiple layers on top of each other. These layers are called OSI layers and each one of them represents an abstraction of that layer and which data given layer contain.

2.1) Ethernet frames

Ethernet frame is a Data Link layer protocol (on OSI layer) that is needed for communication between devices on same local network.

2.2) Internet Protocol version 4 and 6

Internet Protocol is a Network layer protocol (on OSI layer) that is necessary for communication between multiple connected networks (internet). It works by assigning an IP addresses to computers or networks. Simply said when some computer wants to communicate other computer it needs to know its IP address on the network (assuming they are communicating with internet protocol). First version of internet protocol was IPv4, later on an IPv6 was introduced but currently both of them are used, therefore program and most of the internet needs to support both of these standards.

2.3) User Datagram Protocol

UDP is a Transport layer protocol (on OSI layer) designed for fast communication, it does not require prior “handshaking” for communication. UDP consists of data packets that are usually sent with serial numbers that are used for ordering data. Since DNS does not require lots of data to be sent therefore UDP is ideal as it is lightweight and fast.

2.4) Domain name system

DNS is network protocol that translates domain names from human readable form (like “vut.cz”) into a corresponding IP address, therefore allowing easier use of internet. DNS packets are usually send over internet to the DNS servers, which are specialized servers that answer to these requests. These servers are owned by companies such as Google, from which a site provider can buy domain name, after that these DNS servers will translate this domain name into a specific IP address. In this project we well only focus on DNS over UDP with both IPv4 and IPv6.

2.5) PCAP

PCAP is open-source C/C++ library for capturing and monitoring network traffic. It was used in this program for capturing live traffic or opening files that contain captured traffic. It is widely used library and known programs such as Wireshark use it as a backbone.

This program captures said network communication and displays it to the user.

3. Implementation

Application in separated into more logical parts (pairs of source and header files).

3.1) ProgramConfiguration structure (programConfig)

First action done by program is to initialize and configure ProgramConfiguration structure (from now Config). This structure holds all important settings describing program behavior and pointers to all dynamically allocated data. This approach ensures that even if program end is required to shutdown in unexpected way (for example SIGINT before program finishes its task) no memory leaks will happen. This style also pushes programmer forwards reusing already dynamically allocated memory.

3.2) Handling user arguments (argumentHandler)

Argument handler’s purpose is to correctly handle user inputs provided in command line when program is run. Setting all options that change program behavior into Config. This part of program uses C standard library *getopt*.

3.3) Setting packet capturing (pcapHandler)

After Configuration has been updated with user inputs program starts to initialize capturing packets. For capturing packets is used PCAP library. Based on user inputs capturing packets can be initialized to either live capturing of live traffic or to open a stored traffic from a file. As an inspiration for most of this file contents was used article from Tim Carstens from tcpdump.org. There is a known bug where sometimes a configuration fails due to internal errors and program ends with error.

3.4) Capturing / reading packets

Once everything was setup program starts its doing its main purpose, monitoring DNS communication and printing useful information. In main.c a function *packetLooper* is called, where program loops through packets until it found number of packets that provided in arguments. For this is used a *pcap_next_ex* function that waits until a packet is detected. Once detected a packet is send as raw array of bytes into a packetDissector for further processing.

It should be noted that because assignment doesn’t specify how many DNS messages should program capture a default value is one. If user wants to modify this behavior it can be changed with “-n [NUMBER_OR_PACKETS]”, more in usage manual.

3.5) Packet dissecting (packetDissector)

Is the biggest source/header file pair form the whole program. It contains functions for correct dissecting of packets (turning chunk of data it into smaller relevant pieces of information). Program checks for different types of records and prints correct information based on that. Based on provided Configuration will some types of records be stored into a file.

3.6) Buffer structure (buffer)

Buffer could be considered a smaller custom library (set of functions) for working with byte arrays. It provides easy and automatic resizing, adding characters or string to the byte array, comparing Buffers etc… This smaller "library" was written for projects in other course IPK, as this program was similar enough I as an author decided to use this instead of using standard libraries such as *string.h* which would be more appropriate alternative.

3.7) List of Buffer structures (list)

List is an another smaller custom library for structure BufferList which is a double linked list containing Buffers. List is used mainly for storing domain names and translated domains names into IP addresses. Instead of reading from an opened file these records are stored in List and writing to file happens at the end of the program or when error occurs.

4. Using the program

In order to use the program we need its binary, which in this case we can build using Makefile.

Note: it is required to have installed these dependencies on you machine: gcc compiler, libpcap library, and Makefile

Run command in root directory of project (in this directory a README.md or Makefile should be present):

```
$ make
```

After the program is run a binary file in root directory should be created named *dns-monitor*. In order to run and capture communication from live interface you will need to run program with privileged permissions (on unix based system it means to either run program as root or using sudo). If you are reading from file you do not need privileges.

On you user account you can run program like this, lets run it with *-h* or *--help* arguments:

```
$ sudo ./dns-monitor -h
```

Program should print how to use it, let's review it quickly:
In order to run the program you need to specify either an interface or file from which file will be reading:

for reading from a file

```
$ sudo ./dns-monitor -p [FILE_NAME]
```

or to read from live interface. For list the of available interfaces use program with `-o` option<br>

```
$ sudo ./dns-monitor -i [INTERFACE]
```

Note: [FILE_NAME] expects a path to the file and its name, no need to use []

After this you should get your first captured packet. Next you can use *-v* option to get more detailed information about captured packets.

If you want to save all domain names that were captured by program you can use *-d* option and give it name of the file to store this information in.
You can also use *-t* option to capture all translated IP addresses to the domain names, you also need to specify the file:

```
$ sudo ./dns-monitor -i [INTERFACE] -d [DOMAIN_NAME_OUTPUT_FILE]
```

or

```
$ sudo ./dns-monitor -p [FILE] -t [TRANSLATIONS_OUTPUT_FILE]
```

Last parameter is a *-o* parameter that prints all available devices that can be used to live capture packets:

```
$ sudo ./dns-monitor -o
```

5. Testing

Testing was mainly done using *run_tests.sh* shell script that tries most of the inputs with different types of DNS requests. The results are printed to the standard output and file input like domain names are in *build/* directory. It is required to first run a program and then run tests script.

6. Bibliography

PCAP:
https://www.tcpdump.org/pcap.html
https://www.tcpdump.org/manpages/pcap-filter.7.html
https://www.tcpdump.org/manpages/pcap_open_offline.3pcap.html

DNS:
https://datatracker.ietf.org/doc/html/rfc1035
https://datatracker.ietf.org/doc/html/rfc3596

pcapHandler.c was created mostly by following:
https://www.tcpdump.org/pcap.html