Monitoring of DNS communication

Author: Nikita Koliada(xkolia00)

October 30, 2024

Contents

1 Introduction **2**

2 DNS theory **2**
2.1 DNS . . . 2
2.2 DNS types . . . 2

3 Program Usage **3**
3.1 Command Line Arguments . . . 3
3.2 Output format: . . . 3
3.3 Output example: . . . 4

4 Implementation **5**
4.1 Key Functions . . . 5

5 Testing and Results **8**
5.1 Testing Example 1 . . . 8
5.2 Testing Example via pcap file . . . 9
5.3 Testing Example with different types: . . . 10
5.3.1 Example for SOA type: . . . 10
5.3.2 Example for AAAA type: . . . 10
5.3.3 Example for CNAME type: . . . 11
5.3.4 Example for SRV type: . . . 12
5.4 Test Results . . . 12

1 Introduction

The goal of the project is to implement a program called dns-monitor, which will monitor communication on a selected network interface or from a PCAP format file to process DNS packets. The captured packets are processed to extract and log domain names and their translated IP addresses. This documentation will guide through the project's functionality, usage and implementation.

2 DNS theory

2.1 DNS

DNS (Domain Name System)[3]pcap packets follow a specific format that facilitates the querying and responding processes of DNS servers. A DNS packet consists of a header and several sections, including questions, answers, authority, and additional information. The header is 12 bytes long and contains fields such as the identification number, flags, number of questions, number of answers, number of authority records, and number of additional records. The question section specifies the domain name being queried, the query type (e.g., A, AAAA, MX), and the query class (typically IN for Internet). The answer section contains the resource records that provide the requested information, such as the corresponding IP address for a domain name. The authority section indicates which DNS server is authoritative for the domain, while the additional section can provide extra information, such as the IP addresses of the name servers. This structured format enables efficient and standardized communication between DNS clients and servers, ensuring that domain name resolution processes are carried out effectively across the internet.

2.2 DNS types

- **A (Address Record)**: The A record maps a domain name to an IPv4 address, providing the IP address used to locate the domain on the internet.
- **AAAA (IPv6 Address Record)**: Similar to an A record, but it maps a domain name to an IPv6 address, supporting modern internet protocols.
- **MX (Mail Exchange Record)**: The MX record specifies the mail server responsible for receiving emails for a domain, including its priority.
- **CNAME (Canonical Name Record)**: A CNAME record maps one domain name (alias) to another, allowing multiple domain names to point to the same host.
- **SOA (Start of Authority Record)**: The SOA record provides administrative information about the domain, such as the primary DNS server, email of the domain administrator, and timing for DNS zone file refreshes.
- **SRV (Service Record)**: The SRV record defines the location (hostname and port) of servers for specific services.

3 Program Usage

The program can be run in two modes: live capture or via usage of a PCAP file. Can output data in short and in verbose mode as well as can log the domain names with/without their translation.

Compile the project:

```
$ make
```

After the project has been compiled, you can execute commands like:

```
$ ./dns-monitor (-i <interface> | -p <pcapfile>) [-v] [-d <domainsfile>] [-t <translationsfile>]
```

3.1 Command Line Arguments

- `-i <interface> | -p <pcapfile>`: Specifies the network interface to capture packets from or specifies the PCAP file to analyze. Could only be either one of those
- `-v`: Enables verbose mode, printing detailed DNS information.
- `-d <domainsfile>`: Specifies the file for logging extracted domain names.
- `-t <translationsfile>`: Specifies the file for logging domain names and it's translation to IP addresses.

3.2 Output format:

Short mode

```
<YYYY-MM-DD><space><HH:MM:SS><space><source IP><space>-><space><destination IP><space>
(<Q/R - query/response><space><number of records in the Question section>
/<number of records in the Answer section>
/<number of records in the Authority section>
/<number of records in the Additional section>)
```

Verbose mode

```
<Item name>:<space><value>
...
<Item name>:<space><value>
(The item name can be: Timestamp, SrcIP, DstIP, Identifier, Flags)
<empty line>
[Question Section]
<records from the Question section>
(spaces or tabs can be used to separate fields)
<empty line>
[Answer Section] (if it exists, if empty, it can be omitted)
(followed by any records and similarly for other sections)
==================== (serves as a separator for clarity)
```

- **Timestamp** - date and time
- **SrcIP** - source IPv4/6 address from the IP header
- **DstIP** - destination IPv4/6 address from the IP header
- **SrcPort** - source port: `<TCP/UDP>/<port number>`
- **DstPort** - destination port: `<TCP/UDP>/<port number>`
- **Identifier** - query identifier (from the DNS header)
- **Flags** - values of flags QR, OPCODE, AA, TC, RD, RA, AD, CD, RCODE (format: `<uppercase flag name>=<value>`)
- **Section content** - Question, Answer, Authority, Additional

3.3 Output example:

Example command: This captures DNS packets from interface `en0`, prints verbose output, and logs domain names and translations to separate files.

```
make
./dns-monitor -i en0 -v -d domains.txt -t translations.txt
```

Output example in verbose mode:

```
Timestamp: 2024-10-24 11:16:05
SrcIP: 192.0.0.1
DstIP: 8.8.8.8
SrcPort: UDP/53997
DstPort: UDP/53
Identifier: 0x8FB
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]
onedscolprdeus13.eastus.cloudapp.azure.com. IN A
====================
Timestamp: 2024-10-24 11:16:05
SrcIP: 8.8.8.8
DstIP: 192.0.0.1
SrcPort: UDP/53
DstPort: UDP/53997
Identifier: 0x8FB
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
onedscolprdeus13.eastus.cloudapp.azure.com. IN A

[Answer Section]
onedscolprdeus13.eastus.cloudapp.azure.com. 339873 IN A 52.168.117.170
```

The output shows DNS query and response packets in verbose mode

Output example in short mode:

```
make
./dns-monitor -i en0


2024-10-24 12:02:54 192.168.101.160 -> 8.8.8.8 (Q 1/0/0/0)
2024-10-24 12:02:54 8.8.8.8 -> 192.168.101.160 (R 1/8/0/0)
```

The output shows DNS query and response packets in short mode

4 Implementation

This section provides a list of all functions created to extract, parse and generate DNS related data. The main goal of this program is to monitor DNS traffic by capturing and analyzing DNS packets from either a live network interface or a PCAP file, written in C and using key libraries such as [2]pcap.h, netinet/ip.h, netinet/udp.h, arpa.h for extracting DNS info. The program processes and interprets DNS queries and responses, extracting key information such as domain names, record types and IP addresses. It logs domain names and DNS transaction details into files for further analysis. The flow of steps involves capturing packets (capture_packets), filtering DNS traffic (process_dns), parsing DNS data sections (for responses: process_dns_records, for question: process_dns_question), and then logging domain names and its IP address translatation information. Additionally, the program can be run either in verbose mode to provide detailed info or in short mode (by default).

4.1 Key Functions

const char* dns_type_to_str(unsigned short type) .

This function maps DNS record type codes to their corresponding string representations.

Parameters:

- `type`: The DNS record type

Returns:

- A constant string representing the DNS type.

int get_size_of_domain_name(u_char *payload, char *name) .

This function calculates the size of a domain name in a DNS packet, handling both uncompressed and compressed formats.

Parameters:

- `payload`: A pointer to the start of the DNS name section.
- `name`: The buffer to store the domain name.

Returns:

- The total size of the domain name.

int line_exists_in_file(FILE *file, const char *line_to_check) .

This function checks if a specific line exists in file.

Parameters:

- `file`: A pointer to the file to search in.
- `line_to_check`: The string to search for.

Returns:

- `1` if the line exists, otherwise `0`.

void append_domain_if_not_exists(FILE *file, const char *domain) .

Appends a domain name to a file if it does not already exist.

Parameters:

- `file`: The file to write to.
- `domain`: The domain name to append.

void append_transaction_if_not_exists(FILE *file, const char *domain, const char *ip_address)
.

Appends a domain name and paragraph IP address to a file if does not already exist.
Parameters:

- `file`: The file to write to.
- `domain`: The domain name.
- `ip_address`: The translated IP address.

const char* print_timestamp() .
Generates and returns the current timestamp in the format `YYYY-MM-DD HH:MM:SS`.

void print_ip_address(struct ip *ip_header) .
Prints the source and destination IP addresses.

const char* print_dns_flags(uint16_t flags) .
Parses and returns a string representing the DNS flags.

void extract_domain_name(u_char *payload, u_char *msg_buffer, char *domain_name) .
Extracts a domain name from a DNS packet, handling compression if needed.
Parameters:

- `payload`: Pointer to the DNS payload.
- `msg_buffer`: The DNS message buffer.
- `domain_name`: The buffer where the extracted domain name will be stored.

int process_dns_question(char *output_buffer, u_char **payload, u_char *buffer, FILE *domain_file, int verbose) .
Processes the question section of a DNS packet.
Parameters:

- `output_buffer`: The buffer to store the output.
- `payload`: Pointer to the DNS payload.
- `buffer`: The DNS message buffer.
- `domain_file`: The file to append the domain name.
- `verbose`: Verbosity flag.

void process_dns_records(u_char **payload, u_char *buffer, unsigned short count, FILE *translation_file, FILE *domain_file, int verbose) .
Processes DNS records from the answer, authority, and additional sections of a DNS packet.
Parameters:

- `payload`: Pointer to the DNS payload.
- `buffer`: The DNS message buffer.
- `count`: The number of DNS records.
- `translation_file`: The file to append domain-IP records.
- `domain_file`: The file to append domain names.
- `verbose`: Verbosity flag.

void process_dns(const u_char *packet, int packet_size, int verbose, FILE *domain_file, FILE *translation_file) .

Processes a DNS packet by extracting the IP header, UDP header, and DNS sections, then parses them.

Parameters:

- `packet`: The packet data.
- `packet_size`: The size of the packet.
- `verbose`: Verbosity flag.
- `domain_file`: The file to append domain names.
- `translation_file`: The file to append domain-IP records.

void capture_packets(const char *source, int is_pcap_file, int verbose, FILE *domain_file, FILE *translation_file) . Function to set up pcap and then calls packet_handler callback to later deal with dns packets.

5 Testing and Results

The program was tested under various conditions:

- **Live Capture**: Using a physical interface, DNS queries and responses were captured and logged successfully and compared to the same packets in Wireshark [1].
- **PCAP File Analysis**: Several PCAP files containing DNS traffic were processed, and the expected domain names and IP addresses were extracted.

5.1 Testing Example 1

Executing command to create a dns packet:

```
dig @8.8.8.8 isa_example.com A
```

Running the program in short mode:

```
./dns-monitor -i en0
```

Output:

```
2024-10-24 23:08:00 192.168.137.131 -> 8.8.8.8 (Q 1/0/0/1)
2024-10-24 23:08:00 8.8.8.8 -> 192.168.137.131 (R 1/0/1/1)Ï
```

Running the program in verbose mode:

```
./dns-monitor -i en0 -v
```

Output:

```
Timestamp: 2024-10-24 23:24:46
SrcIP: 192.168.137.131
DstIP: 8.8.8.8
SrcPort: UDP/50871
DstPort: UDP/53
Identifier: 0x23F7
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=1, CD=0, RCODE=0

[Question Section]
spotify.com. IN A
====================
Timestamp: 2024-10-24 23:24:46
SrcIP: 8.8.8.8
DstIP: 192.168.137.131
SrcPort: UDP/53
DstPort: UDP/50871
Identifier: 0x23F7
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
spotify.com. IN A

[Answer Section]
spotify.com. 74155 IN A 35.186.224.24
```

if executing with -t or -d parameters, for example:

```
./dns-monitor -i en0 -t translation.txt -d domain.txt
```

The files will have the following result:
For Domain names only: spotify.com.
For Translation file: spotify.com. 35.186.224.24

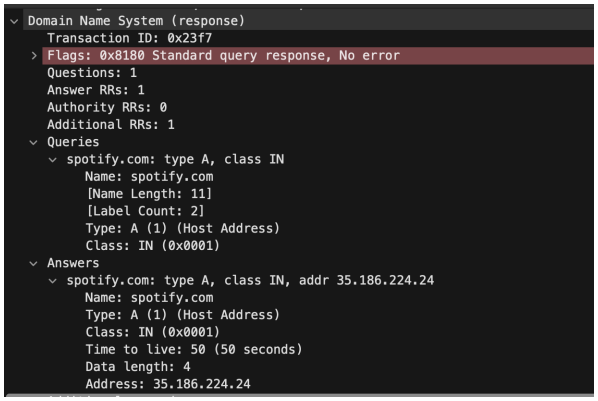

Figure 1: Wireshark example

Example of how it should look like on Wireshark :

5.2 Testing Example via pcap file

We can also use Wireshark for downloading pcap files and test it using the DNS monitor program. Example of an execution with answer, authority and additional sections:

```
./dns-monitor -p test.pcap -t translation.txt -d domain.txt
```

Output:

```
Timestamp: 2024-10-24 23:24:48
SrcIP: 192.168.137.131
DstIP: 192.168.137.1
SrcPort: UDP/7498
DstPort: UDP/53
Identifier: 0xCCB9
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]
optimizationguide-pa.googleapis.com. IN A
====================
Timestamp: 2024-10-24 23:24:48
SrcIP: 192.168.137.1
DstIP: 192.168.137.131
SrcPort: UDP/53
DstPort: UDP/7498
Identifier: 0xCCB9
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
optimizationguide-pa.googleapis.com. IN A

[Answer Section]
optimizationguide-pa.googleapis.com. 4059 IN A 142.251.37.106
optimizationguide-pa.googleapis.com. 4059 IN A 142.251.36.74
optimizationguide-pa.googleapis.com. 14753 IN A 142.251.36.138
optimizationguide-pa.googleapis.com. 14753 IN A 142.251.36.106

[Authority Section]
googleapis.com. 74155 IN NS ns2.google.com.
googleapis.com. 74155 IN NS ns1.google.com.
```

```
googleapis.com. 74155 IN NS ns4.google.com.
googleapis.com. 74155 IN NS ns3.google.com.

[Additional Section]
ns1.google.com. 239339 IN A 216.239.32.10
ns1.google.com. 339873 IN AAAA 2001:4860:4802:32::a
ns2.google.com. 239339 IN A 216.239.34.10
ns2.google.com. 339873 IN AAAA 2001:4860:4802:34::a
ns3.google.com. 239339 IN A 216.239.36.10
ns3.google.com. 339873 IN AAAA 2001:4860:4802:36::a
ns4.google.com. 239339 IN A 216.239.38.10
ns4.google.com. 339990 IN AAAA 2001:4860:4802:38::a
====================
```

5.3 Testing Example with different types:

Here is the test cases for all types that weren't covered over previous test examples. For these test cases I use commands such as dig or nslookup.

5.3.1 Example for SOA type:

Command:

```
dig vut.cz SOA
```

Output:

```
Timestamp: 2024-10-25 09:54:28
SrcIP: 172.20.10.14
DstIP: 172.20.10.1
SrcPort: UDP/53546
DstPort: UDP/53
Identifier: 0x6456
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=1, CD=0, RCODE=0

[Question Section]
vut.cz. IN AAAA
====================
Timestamp: 2024-10-25 09:54:28
SrcIP: 172.20.10.1
DstIP: 172.20.10.14
SrcPort: UDP/53
DstPort: UDP/53546
Identifier: 0x6456
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
vut.cz. IN AAAA

[Authority Section]
vut.cz. 239495 IN SOA rhino.cis.vutbr.cz. hostmaster.vutbr.cz. 2024100801 28800 7200 1814400 300
```

5.3.2 Example for AAAA type:

Command:

```
nslookup -type=AAAA www.microsoft.com
```

Output:

```
Timestamp: 2024-10-25 10:18:22
SrcIP: 172.20.10.14
DstIP: 172.20.10.1
SrcPort: UDP/58823
DstPort: UDP/53
Identifier: 0x1A9D
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]
www.microsoft.com. IN AAAA
====================
Timestamp: 2024-10-25 10:18:23
SrcIP: 172.20.10.1
DstIP: 172.20.10.14
SrcPort: UDP/53
DstPort: UDP/58823
Identifier: 0x1A9D
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
www.microsoft.com. IN AAAA

[Answer Section]
www.microsoft.com. 340029 IN CNAME www.microsoft.com-c-3.edgekey.net.
www.microsoft.com-c-3.edgekey.net. 239495 IN CNAME www.microsoft.com-c-3.edgekey.net.globalredir.akad
www.microsoft.com-c-3.edgekey.net.globalredir.akadns.net. 1220 IN CNAME e13678.dscb.akamaiedge.net.
e13678.dscb.akamaiedge.net. 1220 IN AAAA 2a02:26f0:d8:385::356e
e13678.dscb.akamaiedge.net. 239495 IN AAAA 2a02:26f0:d8:38a::356e
e13678.dscb.akamaiedge.net. 239495 IN AAAA 2a02:26f0:d8:38d::356e
e13678.dscb.akamaiedge.net. 239495 IN AAAA 2a02:26f0:d8:39d::356e
e13678.dscb.akamaiedge.net. 239495 IN AAAA 2a02:26f0:d8:388::356e
```

5.3.3 Example for CNAME type:

Command:

```
nslookup -type=CNAME www.microsoft.com
```

Output:

```
Timestamp: 2024-10-25 10:07:29
SrcIP: 172.20.10.14
DstIP: 172.20.10.1
SrcPort: UDP/50823
DstPort: UDP/53
Identifier: 0x4185
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]
www.microsoft.com. IN CNAME
====================
Timestamp: 2024-10-25 10:07:29
SrcIP: 172.20.10.1
DstIP: 172.20.10.14
SrcPort: UDP/53
DstPort: UDP/50823
Identifier: 0x4185
```

```
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
www.microsoft.com. IN CNAME

[Answer Section]
www.microsoft.com. 1500 IN CNAME www.microsoft.com-c-3.edgekey.net.
```

5.3.4 Example for SRV type:

Command:

```
nslookup -type=SRV _sip._tls.microsoft.com
```

Output:

```
Timestamp: 2024-10-25 10:14:19
SrcIP: 172.20.10.14
DstIP: 172.20.10.1
SrcPort: UDP/58585
DstPort: UDP/53
Identifier: 0x69FB
FLAGS: QR=0, OPCODE=0, AA=0, TC=0, RD=1, RA=0, AD=0, CD=0, RCODE=0

[Question Section]
_sip._tls.microsoft.com. IN SRV
====================
Timestamp: 2024-10-25 10:14:19
SrcIP: 172.20.10.1
DstIP: 172.20.10.14
SrcPort: UDP/53
DstPort: UDP/58585
Identifier: 0x69FB
FLAGS: QR=1, OPCODE=0, AA=0, TC=0, RD=1, RA=1, AD=0, CD=0, RCODE=0

[Question Section]
_sip._tls.microsoft.com. 24667 IN SRV

[Answer Section]
_sip._tls.microsoft.com. 24667 IN SRV 100 1 443 sipdir.online.lync.com.
```

5.4 Test Results

Test cases verified the following:

- Domain names were correctly parsed, even for compressed names.
- A, AAAA, MX, NS, SOA, SRV, and CNAME records were accurately processed.
- Files were properly created and updated, without duplicates.

Figure 2: SOA example

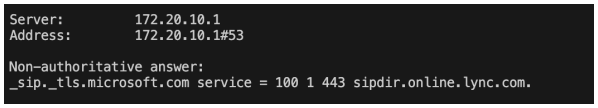

Figure 3: SRV example

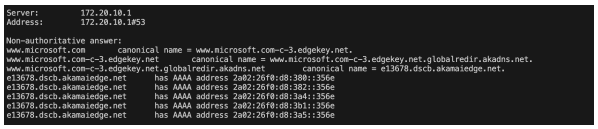

Figure 4: AAAA example

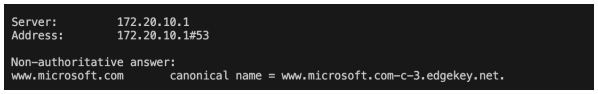

Figure 5: CNAME example

References

[1] Wireshark Foundation. Wireshark documentation. Read on 2024-10-24.

[2] The TCPDump Group. *PCAP Documentation*, 2024. Read on 2024-10-24.

[3] P. Mockapetris. Domain names - implementation and specification. RFC 1035, 1987. Read on 2024-10-24.