Faculty of Information Technology
Brno University of Technology

ISA
Network Applications and Network Administration

Export DNS Information By Protocol Syslog

Attila Lakatos (xlakat01)

November 13, 2018

Contents

1 Abstract

The primary goal of this project is to analyze and print out the description of both incoming and outgoing DNS responses from a valid pcap file. It also allows the user to display DNS packets being transmitted or recieved over a network which the computer is attached. It is also possible to use dnx-export for the specific purpose of summarizing and sending the communications to syslog server. First of all, this document gives a brief introduction about DNS(Domain Name System). After that, we will cover what packet sniffing is as well as how to analyze the data packets that have been captured and analyzed on a network. Then it follows a detailed description about the implementation part describing data parsing on different layers. This will solidify a basic understanding of packet sniffing. The most troublesome parts and difficult steps which made the programming part more complicated will be disscussed in this documentation. Finally, we will encompass the conclusions and the discussion of the results from the project. The authors point of view towards the task is included as well.

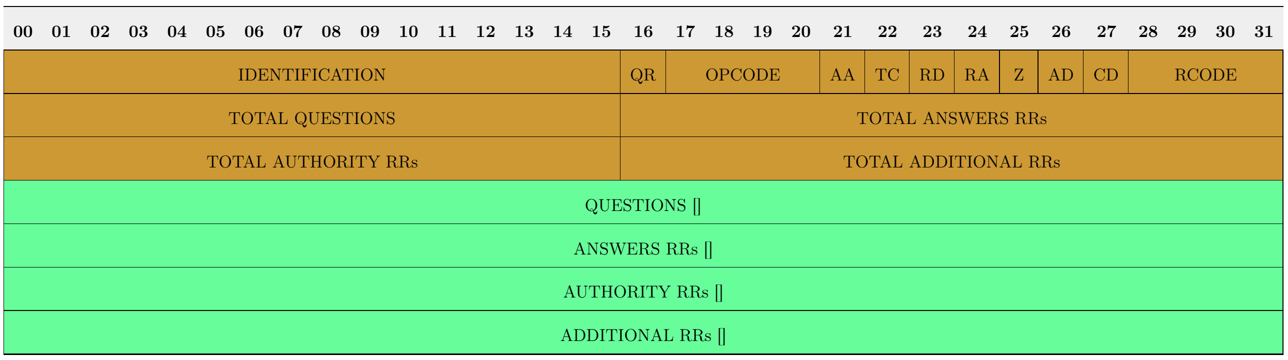
2 Introduction

The Domain Name System (DNS) is an application–layer protocol. This protocol implements the DNS naming service, which is the naming service used on the Internet. This documentation assumes that the reader has some familiarity with network programming. DNS provides a so-called, Name-To-Address Resolution. It is a worldwide hierarchy of computers on the Internet. Although, the basic funcionality of DNS is simplier. Name-to-adress, also known as mapping, is a process of finding the IP adress of a computer in a database by using its hostname.

To use DNS, we send query to a DNS server. This query contains the domain name which we are looking for. DNS server tries to find that domain name's IP adress in its data store. In some cases it finds it, after that it returnes it. If the IP has not been found yet, the server will forward the query to another DNS name server. The process is repeated until the IP is found.[1] DNS is deeply documented in RFC 1035 [2].

2.1 DNS header in nutshell

The following figure describes what kind of informations does DNS header contain. The very first row highlighted by gray color indicates the position of each bit. It is followed by 4 constantly given 32 bits long fields. Each field will be described accurately in the next subsections. Rows having green background color represent fields with variable sizes.



2.2 DNS header section format

1. Identification - This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.

2. Flags

   - QR - Indicates if it is a query or response. In this project we will deal only with responses.
   - Opcode - Query type.
   - AA - Specifies that the responding name server is an authority.
   - TC - 1 indicates that only the first 512 bytes of the reply was returned.
   - RD - Recursive query support is optional.
   - RA - Recursive query support is/is not available in the name server
   - Z - Reserved for future use.
   - AD - It should be set only if all data in the response has been cryptographically verified or otherwise meets the server's local security policy.
   - CD - Checking disabled.
   - Rcode - Response code.

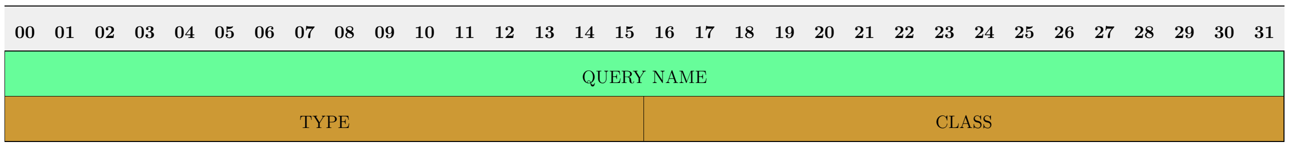
3. Total questions - Number of questions

4. Total answer RRs - Number of answers

5. Total authority RRs - Number of authority RRs

6. Total additional RRs - number of additional RRs

2.3 DNS query section format

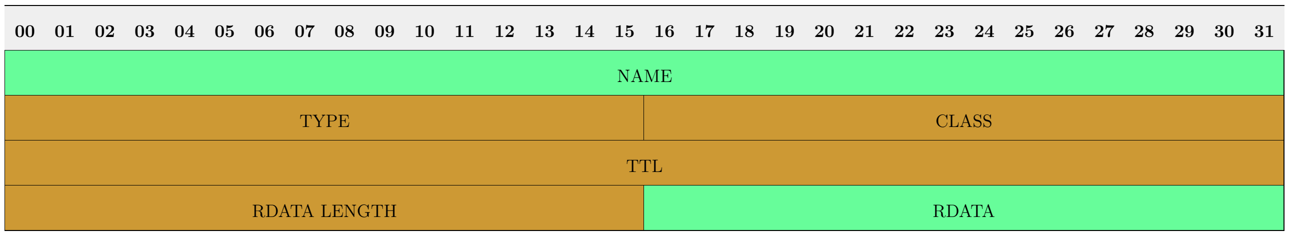


1. Query name - Domain name represented as a sequence of labels. It will be deeply explained in the implementation part.

2. Type - Specifies the type of the query.

3. Class - Specifies the class of the query.

2.4 DNS resource record format



1. Name - Domain name to which this resource record pertains.
2. Type - This field specifies the meaning of the data in the RDATA field.
3. Class - Specifies the class of the data in the RDATA field.
4. TTL - The time interval (in seconds) that the resource record may be cached before it should be discarded.
5. RD length - Specifies the length in octets of the RDATA field.
6. RData - The format of this information varies according to the TYPE and CLASS of the resource record. For more information plese visit RFC 1035[2].

2.5 Information compression

To understand how QNAME is represented(same goes for NAME and RDATA) we need to know how a domain name is stored in a DNS packet. The domain name is represented in the form of labels separeted by dots. It can be represented as either a sequence of labels ending in a zero octet, a pointer or a sequence of labels ending with a pointer. The purpose of using the foregoing representations are needed in order to reduce the size of messages, the domain name system utilizes a compressions scheme which eliminates duplicated domain names in a message. In some cases an entire domain name or a list of labels can be replaced with a specific pointer. As we dive deeper into dns name interpreting, we can see that the first two bits tells us if it is a label or pointer:

- 0b00 - Labels must begin with two zero bits, because labels are restricted to 63 octets or less.
- 0b01 - Reserved for future use.
- 0b10 - Reserved for future use.
- 0b11 - Hexidecimal representation is '0x0c'. This allows a pointer to be distinguished from a label, since a label must begin with 2 zero bits. Remaining bits are being used to calculate where the pointer points to.

3 Implementation

C++ was choosen as the implementation language because it has object-oriented and generic programming features, while also providing facilities for low-lever memory programming. Furthermore, allows to work with strings in a more efficient way. In the next subsections, we will discuss how the program was implemented by progressing gradually from one stage to the next.

As I mentioned earlier, some DNS resource records can vary according to the type and class of its record. These constants can be found in constants.h header file. It also contains error values returned by our program while encountering an invalid state during compilation.

First of all, the whole process starts with parsing input arguments. After that, each parameter is stored in a structure for further application. Functions connected to argument checking take place in argparer.cpp. In case of an error occured during compilation a brief message is displayed by a program that utilizes a command-line interface for execution. For further information man(dns-export) is available too.

At this point, program will continue in one of the following ways. A valid file has been provided by a user to parse packets from it or an interface flag was specified which causes it to read packets from a network interface rather than read from a saved packet. Reading packets from a network interface may require that you have special privileges; see the pcap man page for details. Reading a saved packet file doesn't require special privileges. These two methods share same algorithms, they only differ at the begining where I define either an online[1] or offline[2] method. Fortunately, pcap header file provides us some useful functions to parse data from dns packets. I will be sure to describe concepts in greater detail.

We begin by identifying which interface we want to sniff on. We can either define an interface as a string or we can sniff on all interfaces by means of 'ANY'. Next, pcap needs to be initialized. We differentiate between devices using file handles, just like an ordinary ones for file handling. The above listed steps only refered to live sniffing. Then, we must create a rule set not to parse unneccssary packets. This can be achieved by setting port to 53. Compiling and aplying is also required. The rule set is kept in a string, and is converted into a format that pcap can read (hence compiling it). Finally, we tell pcap to enter it's primary execution loop. In this state, pcap waits until it has received however many packets we want it to. Every time it gets a new packet in, it calls another function that we have already defined.

From now on, things will constantly get more and more complicated. First of all, pcap_datalink(p) returns the link-layer header type for the live capture or "savefile" specified by p. We only deal with ethernet and linux cooked capture. It gives us information about which internet protocol version(IPv4 or IPv6) does it support. After successfully parsing link-layer header, we can move on to internet protocol version. Size of IPv4 is constantly defined, while IPv4 has a special field containg additional information connected to its size. Penultimate task is to successfully identify and handle the TCP/UDP layer. Most likely DNS packets will arive as UDP ones but in some cases, such as longer and more detailed answers can appear. They can be handled via TCP. UDP header size is set in advance, contrary to TCP where it need to be read from a special field called th_off(the length of the TCP header is always a multiple of 32 bits).

At the current state we have a pointer which tells us where does DNS query/response start. By the help of transaction ID we can identify which query is connected to which response. This property is properly used while tracking down TCP packets. DNS packets using TCP protocol contain an additional 2 byte length field. There are some obstacles between us and DNS answers rigth now, to get right there we have to skip Transaction ID, Flags, Questions, Answers, Authority RRs and Additional RRs; as I mentioned earlier their size is constantly defined. Only fields that remain are queries, for this purpose I have defined a so-called jumpToDnsAnswers() function, which skips the whole unnecessary

[1]online: read packets from network interface

[2]offline: read from a saved packet file

content. More detailed description by using pseudocode is also availabe in section 4.

As the project task requires, the very first indispensable information containing a name needs to be analysed and stored for further processing. It is achieved when dnsNameToString() returnes a valid name, it also ensures pointer will point to the next field. The type field indicates how many additional fields does DNS response contain. This is solved by calling parseDNSdata() function which returnes a string containing additional resource record informations. Every single information related to resource records is being stored in a C++ list of strings for further processing. More detailed informations about above listed functions can be found in section 4.

3.1 Signal handling

Signals are the interrupts delivered to a process by the operating system which can terminate a program prematurely or can affect their interior behavior in an elegent way. If dns-export encounters a SIGUSR1 signal, typically generated by the user calls printOutStatistics() function which prints out data according to every single gathered information till now.

When a user specifies a syslog server, gathered information is also transferred to syslog server via UDP protocol. It consists of two steps, creating syslog header format plus creating syslog message format. The above stated functions are defined in syslog.cpp. The alarm() function causes the system to generate a SIGALRM signal for the process after the number of realtime seconds specified by seconds have elapsed. SIGALRM signals are handled by sendToSyslog() functions, which casues an immediate data transfer to the specified syslog server.

3.2 Limitations

It is hard to define what parts of the assignment have been left out. Reassambled TCP packet analysing, parsing and information gathering are not part of final program, unlike simplier TCP packets.

3.3 Testing

Thanks to available pcap files provided by Ing. Petr Matoušek, Ph.D., M.A. via information system I was able to test the main parts of DNS pacet data parsing. There are 2 tools which need to be mentioned. Wireshark is a network protocol-analyzer, it helped me to see what was happening on a certain network or in a valid pcap file. It has a bunch of features, including packet observation, filtering and analysing. DIG(domain information gropper), an administration command-line tool was also used for querying Domain Name System servers.

4 Algorithms

Algorithm 1 Skip DNS query section

```
procedure jumpToDnsAnswers(label)
    if label is NULL then
        return NULL
    end if
    if label is a pointer then
        return label+2
    end if
    while label is not char with 0 ASCII CODE do
        Increase label pointer
    end while
end procedure
return label+1
```

Algorithm 2 DNS name to string representation

```
procedure dnsNameToString(label, payload, end)
    $string \leftarrow$ ""
    while adress of label < adress of end AND label is not a char with 0 ASCII CODE do
        if label is a pointer then
            $tmp \leftarrow payload + characterWhereLabelPoints$
            while adress of tmp < adress of end AND tmp is not a char with 0 ASCII CODE do
                if tmp is a pointer then
                    $tmp \leftarrow payload + characterWhereLabelPoints$
                end if
                $len \leftarrow label$
                $tmp \leftarrow tmp + 1$
                $string \leftarrow$ len number of characters
                $string \leftarrow string +$ '.'
            end while
            $label \leftarrow label + 1$
        else
            $len \leftarrow label$
            $label \leftarrow label + 1$
            $string \leftarrow$ len number of characters
            $string \leftarrow string +$ '.'
        end if
    end while
    return string
end procedure
```

5 Conclusion

A couple of scientific reports have been surveyed, mainly connected with packet parsing provided by different sources on the internet. The most troublesome part was to precisely interpret the content of DNS labels and pointers due to lack of available materials connected to this topic. I have to admit this project belongs to the most interesting ones which I had to deal with in my studies at VUT FIT. As I mentioned earlier in subsection 3.2, some objectives, such as parsing reassambled TCP packets have been left out due to lack of time.

References

[1] Oracle Corporation and/or its affiliates: *System Administration Guide: Naming and Directory Services*. 2010. [Online; Accessed: 2018-10-11].
Retrieved from: `https://docs.oracle.com/cd/E19683-01/806-4077/6jd6blbbp/index.html`

[2] P. Mockapetris: *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*. 1987. [Online; Accessed: 2018-09-29].
Retrieved from: `https://www.ietf.org/rfc/rfc1035.txt`