Brno University of Technology

Faculty of Information Technology

Network Applications and Network Administration

Programming of network service (2.variant)

Export DNS information using Syslog

November 19, 2018

Stupinský Šimon, (xstupi00)

Contents

Chapter 1

Introduction

This documentation is focused on the description design and implementation of **DNS Export** tool. The first main target of this project was to get acquainted with the *DNS System* and problems, which arise at implementation the application connected with it. *DNS System* use millions of people around the world daily and not just because of it, is its role in the Internet important. The second target in this project was to get acquainted with the *Syslog* protocol.

The primary tasks of *DNS Export* tool is monitoring the *DNS traffic* and reporting the statistics from this traffic. The application can run in two modes - **online** or **offline**. **Online** mode represents the monitoring *DNS traffic* on the given *network interface*. In the contrary, the **offline** mode represents the processing of given *pcap file* in which can be potentially caught *DNS traffic*. In both modes is option to send the obtained statistics to concrete *Syslog* server regularly in intervals. The another details about using of applications will be introduced in Chapter 5. We will mention the interesting part of the implementation, introducing the reasons of some solutions of the problems.

Chapter 2 deals with the *Domain Name System* (DNS) architecture. We will get acquainted with the basic building elements of the system, the way of data storage and access to them in the Chapter 3. In the Chapter 4 will be describe the individual *Resource Record* which are part of *DNS System*, whether of its extension *DNSSEC*.

Chapter 2

DNS Architecture

2.1 Overview

The *domain name space*, *DNS Servers* and *resolvers* represents three main components of which the *DNS System* is composed. The *domain name space* contains the arrangement of data that are stored in the *DNS Systems* and allows access to this data. *DNS Servers* stores data in its local databases with use the specific format, which is defined by individual structures in *DNS Architecture*. *Resolver* is used to access to data, that are stored in the DNS System. In the following sections will be described the individual mentioned components of *DNS System* in more details [3].

2.2 Domain Name Space

The *domain name space* of *DNS System* is composed to levels. The names of individuals levels, and the portion of the *fully qualified domain name* (FQDN) that it represent, are important to know when configuring it. The count of the *domain names* and its *addresses* is a quite high. To achieve the efficiently saving and search is the logical space of all domain names stored in the *DNS System*, in the special way, to following levels.

- **Top-Level domain** - the name of two or three letters used to indicate the country, region or the type of organization, e.g. **„.com“**
- **Second Level domain** - the names of variable length registered to an individual or organization to use on the Internet, e.g. **„microsoft.com“**
- **Sub-domain** - additional names that an organization can create that are derived from the registered *second-level* domain name, e.g. **„example.microsoft.com“**

2.3 DNS Servers

DNS Server is an application that manages the data stored in the *Domain Name Space*. This space is divided into some special zones, which are subsequently relocated into individual *DNS Servers*. Its main task is answers to *DNS Queries*, which are directed on the concretely items stored in the competent *DNS* database. The data itself are stored in the form of *DNS Records*, respectively *Resource Records*.

These records are stored in the *local file* or in another case, will read from the other *DNS server* with using transformation of the zone. The information that server manages and to which is responsible are called *authoritative.* According to specification of the server are defined two basic types of *DNS Servers* - *primary* and *secondary.*

Primary Server DNS, so called *master* or *primary name-server*, contains the fully records about domains which manages. These *records* are storing locally in the files, which are stored in the database of competent *server.* Server provides *authoritative* (i.e always exactly) response to these domains. For every domains can exist just **one** primary name-server.

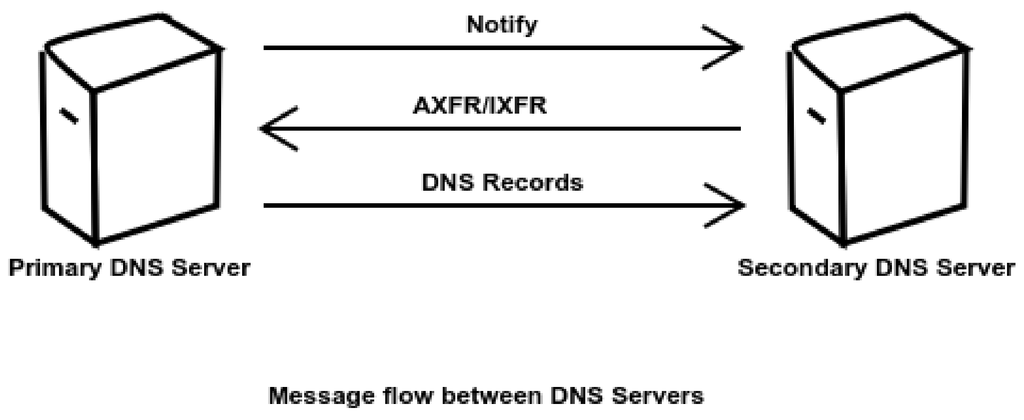

Figure 2.1: The DNS system provides the Notify3 feature. With this feature a primary DNS provider can notify the secondary providers that the records have changed. After receiving the Notify message, secondary servers can use AXFR4 or IXFR5 query type to fetch the zone record [6].

Secondary Server DNS, so called *caching-only* name-server, obtains required data from the *primary servers.* The file, which contains the database of concrete domains (subdomains) is called *zone file.* The process of transport zone file from the *primary* server to *secondary* server is called *zone transfer.*

2.4 Resolver

The *Resolver* is the client program which executing the *DNS Queries* to data stored in the *DNS System.* The majority of the users program, which need information from *DNS System* using to access to this data just *Resolver.* The main tasks of the *Resolver* are:

- send *DNS Queries* to *DNS Servers*
- interpretation of *DNS Responses* from the *DNS Servers* (e.g. check error messages, receiving records)
- pass the information to the user program that requested data

A *Resolver* must be able to access at least to one *DNS Server.* Subsequently gets *DNS Response* directly from the *DNS Server* or in another situation *DNS Server* returns the link to the next server, where the searched information can be saved. *Resolver* can then forward *DNS Query* to obtained servers.

Chapter 3

DNS Protocol

3.1 DNS Message

All communications inside of the *DNS protocol* are carried in the single format of *DNS Message*. The top level format of message is divided into five section shown below:

Header	**Question**	**Answer**	**Authority**	**Additional**

Table 3.1: Format of DNS Messages

Gradually will be described the individual parts at the top level of *DNS Message*. The *Header* section is always present in *DNS Message*. The header part includes fields that specify which of the remaining sections would be present, respectively which section **must be** present. Also specify whether the actual *DNS Message* is a *DNS Query* or *DNS Response* and some other fields mainly to manage of communication.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Identification Number															
QR	**OPCode**				**AA**	**TC**	**RD**	**RA**	**Z**			**RCode**			
QDCount															
ANCount															
NSCount															
ARCount															

Table 3.2: Format of Header of the DNS Message

In the frame of this project is desirable pay more attention to following fields:

- **Identification Number** - *identifier* assigned by the program that generates any kind of *DNS Query*. This *identifier* is copied to corresponding *DNS Reply*. Traditionally is used by the requester to match up replies to outstanding queries. In our case it will serve for paring the individual *DNS Queries* and *DNS Response* according to its *Identification Number*.

- **QR - Query/Response** - one bit flag that specifies whether this message is a *DNS Query* (0), or a *DNS Response* (1). The same purpose of use in our case.

- **RCode - Response Code** - 4 bit field is set as part of response. Can be set on the differently values, but in the context of this project is important only one. At processing of *DNS Response* **must be** value of this field equal to **0** (`RCode == 0`), whats signed *no error condition.*

- **QDCount - Queries count** - 16 bit unsigned integer specifying the number of entries in the question section. At processing of *DNS Queries* **must be** greater than **0** for accepting the given *DNS Query.* At processing of *DNS Response* serves to indicates of count *DNS Queries* in Queries field to its skipping.

- **ANCount - Answers count** - 16 bit unsigned integer specifying the number of entries in the answer section. At processing of *DNS Responses* **must be** greater than **0** for accepting the given **DNS Response**. Also server for indicates the count of **Resource Records** in answer section.

In the next part which follows will be introduced the *Question* section. The *Question* section is used to carry the „question“ in most queries, i.e., the parameters that define what is being asked. The section contains *QDCOUNT* (usually 1) entries, each of the following format:

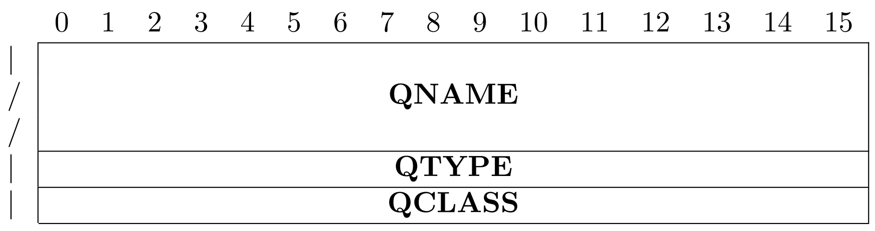

Table 3.3: Question section format

Where the individual field have the following meaning:

- **QNAME - Query Name** - domain name represented as a sequence of *labels.* Each label consists of a length octet followed by that number of octets. If the length octet is equal zero, the *domain name* terminates for the null label of the root. This field from the *Question section* is skipped by this project, because it have no use.

- **QTYPE - Query Type** - two octet code which specifies the type of the *Query.* This field can includes all codes valid for *TYPE* field. Some more general codes can match more than one type of *Resource Record*, which will be introduced in Section 4.

- **QCLASS - Query Class** - two octet code that specifies the *class* of the *Query.* For example, the *QCLASS* field is **IN** for the **Internet**.

The next remaining fields from the format of *DNS* Message represents the variable number of *Resource Records* in the relevant *Records* arrays. Specifically, it is the *Answer*, *Authority*, and *Additional* sections. Each *Resource Record* has the following format:
Where the individual field have the following meaning (*TTL* have no use in this project):

- **NAME** - domain name to which this resource record pertains. This field will be the **first** part of *stats listing* in this project.

- **TYPE** - two octets containing one of the *RR type* codes. This field specifies the meaning of the data in the *RDATA* field. This filed will be the **second** part of *stats listing* in this project.

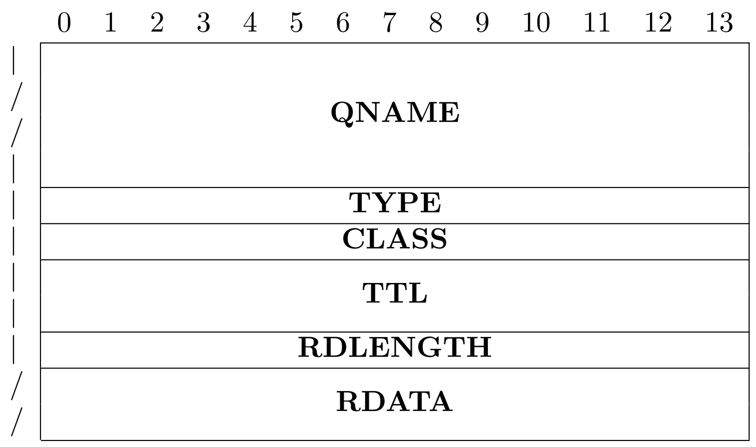

Table 3.4: Resource record format

- **RDLENGTH - Resource Data Length** - an unsigned 16 bit integer that specifies the *length* in octets of the *RDATA field*
- **RDATA - Resource Data** - a variable length string of octets that describes the *resource.* The format of this information varies according to the *TYPE* and *CLASS* of the *Resource Record.* Also this field will be the last part of *stats listing* in thsi project.
- **example**: if the *TYPE* is **A** and the *CLASS* is **IN**, the *RDATA* field is a **4 octet ARPA Internet address**.

All concepts of the tables and details of individuals fields were taken from [**?**].

3.2 DNS Packet Compression

The text data compression is occurs at creating the DNS packets. This activity is useful, because some of the chains in the DNS packet repeat such domain name, etc. Therefore, using the pointer to the first occurrence of a string instead of the name. Each domains name is composed from the sequence of chars contains the domain name (i.e. `"merlin"`, `"fit"`, `"vutbr"`, `"cz"`).

The domain name is stored like as the sequence of pairs (length/value). Each pairs contains length of domains and domains name. This sequence of pairs is ended by zero octet. For example, the domain name `"eva.fit.vutbr.cz"` will be written in the packet like as string of chars (`3 eva 3 fit 5 vutbr 2 cz 0`), where the first byte signed the length of string, which will be following. The sequence is ended by zero char, which is actually the name of the root node in the DNS name tree.

Use of pointers direct to significant reduction of size of the packet. Repeated the occurrences of parts of the domain name in the packet will be changing by pointers on the first occurrences. The array Offset determines the distance from the beginning of the DNS packet, i.e. from the first occurrences octet DNS.

Because the first two bytes of length are used to distinguish of string from the pointer, for the length will be using the following six bit. Because the string contains the domain name can be the maximal length equal to $2^6 - 1$, i.e. 63 bytes. The standard also states that the maximum length of the domain name (concatenation of all domains) is 255 bytes

Chapter 4

Resource Records

DNS Records, so called *Resource records* or *RR*, are used to store information in *DNS* data space. The *records* itself are stored in the text format in *zone files* on responsible *DNS servers*. In the *DNS Queries*, respectively *DNS Response* are defined by *QTYPE*, respectively by *TYPE* field. This section relies on [1]

Between the most useful record belongs the **A** records that map the domain name to the IP address, this process is so called *direct mapping*, and the **PTRs** record for the reverse mapping. Besides these two records are still on look others records, that will be used in this project and it will be gradually introduce:

4.1 General DNS Resource Records

- **A - IPv4 address record - 1** - include direct mapping of *domain name* to *IPv4* addresses. It is the only record that contains the IP address on the right side. Hosts that have multiple Internet addresses will have multiple **A** records. Records itself contains the **32-bit** IPv4 address.

- **NS - Name Server - 2** - Defines the *authoritative name server(s)* for the domain (defined by the SOA record) or the subdomain. Using these records is building the hierarchical DNS structure. This record that is located in the given *DNS* tree node contains pointers to the node followers.

- **CNAME - Canonical Name - 5** - assigns to the alias the canonical (official) name of the computer. Each time when the *DNS server* finds a *CNAME* record in the search, it replaces the canonical domain name and continues in searching. The alias defined by *CNAME* record **must** never stand on the right side of the record. All other records **must** be mapped to the canonical (official) name, not the alias.

- **PTR - Pointer - 12** - IP address (IPv4 or IPv6) to host, so performs reverse mapping. It means that it converts the numeric *IP addresses* to the *domain name*. Reverse addresses are stored in a special domains: „`in-addr.arpa."` for **IPv4** and „`ip6.arpa."` for **IPv6**. These records are simple data, and don't imply any special processing similar to that performed by C*CNAME*, which identifies aliase. *PTR* records cause no additional section processing.

- **SOA - Start of Authority - 6** - Defines the *zone name*, an *e-mail contact* and *various time* and *refresh values* applicable to the zone. SOA records cause no additional section processing. **NS** resource records are required because *DNS Queries* respond with an authority section listing all the authoritative name servers and for queries to sub-domains where they are required to allow referral to take place. Below is viewed the structure of *SOA* record. The purpose of individual fields is described in the project code documentation.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
/ /	**MNAME**															
/	**RNAME**															
| |	**SERIAL**															
| |	**REFRESH**															
| |	**RETRY**															
| |	**EXPIRE**															
| |	**MINIMUM**															

Table 4.1: SOA Resource Record

- **MX - Mail Exchanger - 15** - A preference value and the host name for a mail server/exchanger that will service this zone. **MX** records cause type **A** additional section processing for the host specified by *EXCHANGE.* The number of **MX** records is not limited and **may** be defined any number of these records. In the case when the *mail server* lies within domain (in-zone) it requires **A record**. The purpose of both fields is described in the project code documentation.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
|	**PREFERENCE**															
/ /	**EXCHANGE**															

Table 4.2: MX Resource Record

- **TXT - Text - 16** - Text information associated with a name. **TXT** Resource Records are used to hold descriptive text that provides the ability to associate some arbitrary and in-formatted text with a host or other name. The semantics of the text depends on the domain where it is found. Contains one or more character-string.

- **AAAA - IPv6 Address record - 28** - maps domain addresses to IP addresses of version 6 (**IPv6**). **AAAA** resource records type is a record specific to the Internet class that stored a single **IPv6** address. A 128-bit **IPv6** address is encoded in the data portion of this resource record in network byte order. The textual representation of the data portion of the **AAAA** resource record used in a master database file is the textual representation of an *IPv6 address.*

- **SPF - Sender Policy Framework - 99** - Typically used in conjuction with a *TXT RR* containing the same information and, apart from the *RR* type, the same format. Defines the servers which are authorized to send mail for a domain. Its primary function is to prevent identity theft by spammers. The **SPF** information **must** be defined using a standard *TXT RR.*

- **SRV - Service - 33** - identifies the host(s) that will support a particular service. The *MX* RR is a specialised example of service discovery while the **SRV** RR is a general purpose RR to discover any service and allows control over prioritization of delivery and usage. The theory behind **SRV** is: given a domain name **„example.com“**, and a service name **„http“ (web)** which runs on a protocol **TCP**, then the domain name stored at **SRV** RR will look as follows: `„_http._tcp.example.com."`. After the field with whole *domain name* „part“ follows the **SRV** structure, the purpose of individual fields is described in the project code documentation:

	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
|	**PRIORITY**
|	**WEIGHT**
|	**PORT**
/ /	**TARGET**

Table 4.3: SRV Resource Record

4.2 DNSSEC Resource Records

Standard **DNSSEC** (*DNS Security Extension*) defining extension of DNS protocol for security transport of the data in the *DNS System* with using the *asymmetric cryptography* with using two type of keys - **private** and **public**. The *private* key serves to signing and the *public* key for verifying the signatures. The *private* key is stored in a safe place and the *public* key is freely available.

DNSSEC use the new *Resource Records* in *DNS System*. The all these *Resource Records* are used to signing of the *zones* with *DNSSEC* mechanism. The signed *zone* means that the *zone file* contains except for all *Resource Records* of this zone (*A*, *CNAME*, *PTR*, etc.) the electronic signature for every these records. In the next sections will shortly described the **DNSSEC** *Resource Records* which are the processing by this project. The format of the these *DNSSEC Resource Records* in the statistics field of this application is defined by [4] and [2], for each *Resource Record* by section *Presentation Format.*

- **DNSKEY - DNS Key Record - 48** - contain the *public* key (of an asymmetric encryption algorithm) used in zone signing operations. *Public* keys used for other functions are defined using a *KEY RR. DNSKEY RRs* may be either a *Zone Signing Key* (ZSK) or a *Key Signing Key* (KSK). The *DNSKEY RR* is created using the `dnssec-keygen` utility supplied with `BIND`. The next part of record except of *key* is also the type of key and algorithm for verifying and the another fields, see table below. The purpose of individual fields is described in the project code documentation.

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31
|	**Flags**	**Protocol**	**Algorithm**
/ / /	**Public Key**		

Table 4.4: DNSKEY Resource Record

- **RRSIG - Resource Record Signature - 46** - Each *RRset* in a signed zone will have an **RRSIG** *Resource Records* containing a digest of the *RRset* create using a *DNSKEY RR Zone Signing Key* (ZSK). **RRSIG** *Resource Records* are unique in that they do not form *RRsets* - were this not so recursive signing would occcur, what would be a danger. The signature **RRSIG** does not apply to individual records but to set of *RRs*, which have the same *domain name* (owner), *class*, *type* and *TTL*. From these data and contains of these records will be computing the signature. **RRSIG** RR are automatically created using the `dnssec-signzone` utility supplied with `BIND`. The whole structure **RRSIG** record is the following (the details of the meaning of the individual items are described in the project code documentation):

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31
|	**Type Covered**	**Algorithm**	**Labels**
|	**Original TTL**		
|	**Signature Expiration**		
|	**Signature Inception**		
|	**Key Tag**	**Signer's Name**	
/ / /	**Signer's Name**		
/ / /	**Signature**		

Table 4.5: RRSIG Resource Record

- **NSEC - Next-Secure Record - 47** - The **NSEC RR** points to the next valid name in the *zone file* and is used to provide proof of non-existence of any name within a zone. The last **NSEC** in zone will point back to the zone root or apex. So, this record contains two important information: the next *domain name* of record in the *zone* and the *DNS Resource Records*, which are connected with this domain name. Use of **NSEC** records requires sorting the individual *DNS Records* according to domain names. The sorting is **logic**, it means that is not related with the actual location of records in the *zone file*. **NSEC** RRs are created using the `dnssec-signzone` utility supplied with `BIND`. The details about the **NSEC** structure are described in the project code documentation.

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31
/ /	**Next Domain Name**		
/ /	**Type Bit Maps**		

Table 4.6: NSEC Resource Record

- **DS - Delegation Signed - 43** - it appears ath the point of delegation in the parent zone and contains a digest of the *DNSKEY RR* used as either a *Zone Signing Key* (ZSK) or a *Key Signing Key* (KSK). It is used to authenticate the chain of trust from the parent to the child zone. **DS** and *DNSKEY* records have the same owner (i.e. they contain the same domain name), but they are stored in different places. The **DS** record is stored in the parent zone. On the contrary the *DNSKEY* RRs are stored in the child's zone. The **DS** record contains the following items (the details of the meaning of the individual items are described in the project code documentation):

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31
|	**Key Tag**	**Algorithm**	**Digest Type**
/ / /	**Digest**		

Table 4.7: DS Resource Record

- **NSEC3 - Next-Secure Record version 3 - 50** - provides authenticated denial of existence for *DNS Resource Record Sets*. The **NSEC3** RR list *RR* types present at the original owner name of the **NSEC3** RR. It includes the next hashed owner name in the hash order of the zone. The complete set of **NSEC3** RRs in a zone indicates which RRSets exist for the original owner name of the RR and form a chain of hashed owner names in the zone. This information is used to provide authenticated denail of existence for *DNS data*. The **NSEC3** RR indicates which *hash function* is used to construct the hash, which *salt* is used, and how many iteration of the hash function are performed over the original owner name. This items are again described in the project code documentation and represented in the structure below:

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31	
|	**Hash Algorithm**	**Flags**	**Iterations**	
|	**Salt Length**	**Salt**		/ /
|	**Hash Length**	**Next Hashed Owner Name**		/ /
/ /	**Type Bit Maps**			/ /

Table 4.8: NSEC3 Resource Record

- **NSEC3PARAM - NSEC3 parameters - 51** - contains the *NSEC3* parameters (*hash algorithm*, *flags*, *iterations* and *salt*) needed by authoritative servers to calculate hashed owner names. The presence of a **NSEC3PARAM** RR at a zone apex indicates that the specified parameters may be used by authoritative servers to choose an appropriate set of *NSEC* RRs for negative responses. This record is not used by validators or resolvers. The details about the individuals items of **NSEC3PARAM** are described in project code documentation and its structures is showe below:

	0 1 . . . 14 15	16 17 . . . 22 23	24 25 . . . 30 31	
|	**Hash Algorithm**	**Flags**	**Iterations**	
|	**Salt Length**	**Salt**		/ /

Table 4.9: NSEC3PARAM Resource Record

Chapter 5

Implementation

In this chapter will be introduce implementation above mentioned *DNS Export Tool*. The theoretical foundation was laid in the previous chapters. The main task of this project was creating the tool, which will be monitoring *DNS Traffic* and logging processed statistics. Better said, the application will catching the incoming *DNS Queries* and *DNS Response* and creating the statics from the individual *Resource Records*.

5.1 Specification of the Arguments

Th arguments entered on the command line are processed by `getopt` utility[1]. In this application are acceptable five arguments, which the following meaning:

- **–help / -h** - printing of the manual page. Also activated in the case of wrong combinations of arguments.
- **–interface / -i - „any“** - online mode - sniffing on the interface. Can be specified **only once**, else error will be occurred. Disabled combination with `-r` parameter.
- **–pcap_file / -r** - offline mode - pcap file processing. Can be specified **more than once**, then will be processed all entered files gradually. After the processing of all pcap files will send the statistics to syslog server, if was entered. The statistics are computing from all entered files together. Appropriate for obtaining statistics from the several files concurrently. Disabled combination with `-i` parameter and `-t` parameter.
- **–syslog / -s** - syslog servers for sending the obtained statistics. Can be specified **more than once**, then will be statistics send to each server individually. Application i obtaining the *addresses* of given server using by `getaddrinfo`[2] and then it tests all addresses unless all the messages are sent successfully. In the *offline* mode, when the syslog server was not entered, then the statistics will be print on the standard output after the processing of all entered pcap files.
- **–time / -t - 60 seconds** - interval for sending the statistics to syslog messages in **seconds**. Can be specified **only once**, else error will be occurred. They must be given only in the present of `-s` parameter and must not be given in present of `-r` parameter.

[1] http://man7.org/linux/man-pages/man3/getopt.3.html
[2] http://man7.org/linux/man-pages/man3/getaddrinfo.3.html

5.1.1 Examples of the Running of Application

1. Sniffing on the **„any“** interface. No sending to syslog server. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export`

2. Sniffing on the **„any“** interface. No sending to syslog server. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export --interface=any`

3. Sniffing on the **„any“** interface. Sending statistics to `localhost` syslog server each **60 seconds**. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export --interface=any --syslog=localhot`

4. Sniffing on the **„any“** interface. Sending statistics to `localhost` syslog server each **60 seconds**. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export -s localhost`

5. Sniffing on the **„any“** interface. Sending statistics to `localhost` syslog server each **10 seconds**. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export -s localhost -t 10`

6. Sniffing on the **„eth1“** interface. Sending statistics to `localhost` syslog server each **10 seconds**. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export -i eth0 -s localhost -t 10`

7. Sniffing on the **„any“** interface. Sending statistics to `localhost` syslog server and `google.com` server each **10 seconds**. Printing the statistics at receiving the *SIGUSR1* signal on standard output.

- `$ ./dns-export --syslog=localhost --syslog=google.com --time=10`

8.Processing of **„pcap_files/dns.pcap“** file. Printing the statistics after the processing on standard output. No sending to syslog server.

- `$ ./dns-export -r pcap_file/dns.pcap`

9. Processing of **„pcap_files/dns.pcap“** file. Sending statistics to `localhost` syslog server after the processing. No printing on standard output.

- `$ ./dns-export --pcap_file=pcap_files/dns.pcap --syslog=localhost`

10. Processing of **„dns.pcap“** file and **„dns6.pcap“** file. Sending statistics to `localhost` syslog server after the processing of both files. No printing on standard output.

- `$ ./dns-export -r school.pcap -r dns6.pcap -s localhost`

5.1.2 Examples of Invalid Combinations of Arguments

1. Missing the syslog server to send the stastics. Alone `-t` parameter not make any meaning.

   - `$ ./dns-export --time=10`

2. Missing the syslog server to send the stastics. Alone `-t` parameter not make any meaning.

   - `$ ./dns-export --interface=eth0 --time=10`

3.Alone `-t` parameter not make any meaning, nonsense combination.

   - `$ ./dns-export --pcap_file=dns.pcap --time=10`

4. Parameter `-t` not make any meaning at processing of pcap files, the statistics will sending after the its processing.

   - `$ ./dns-export --pcap_file=dns.pcap --syslog=localhost --time=10`

5.2 Processing of Packets

On the base of selected mode (*online* or *offline*) is opened the session in the promiscuous mode. Subsequently, will be set the filter to the packet capture handle to monitoring the *DNS Traffic* - „`port 53`". In next phase will be processed the packets that will be caught by created packet capture handler. These introductory actions are executing with using the `pcap` built-in library[3]. In the next subsections we focused on the processing of packets on the individual layers of *TCP/IP Model*.

5.2.1 Datalink Layer

Processing at the lowest layer of the *TCP/IP* model is based on the *type* of this layer, respectively on length of its header. After the creating of packet capture handler is used the built-in function `pcap_datalink`[4], that returns the link-layer header type. This type is send to switch of all supported link-layer header types and it returns the its length. With obtained length can be skip the whole link-layer up to begin of the *network layer*. Subsequently on the base of the **first** byte is distinguished the type of the header on the *network layer*. All supported *link-layer* header types you can see in the relevant switch of these types, that was taken from *nmap tool*[5].

5.2.2 Network Layer

As was said above, on the base of the first byte at the begin of this layer is decoded the type of header. This byte is represented by the `VERSION` field in the structures of headers in this layer. The acceptable types in the frame of this application are **IPv4** and **IPv6** Headers.

In the case, when the header is of **IPv4** type the processing is as follows. In the first step is check whether the value of *IP_HEADER_LENGTH* is in the acceptable range, so

[3] http://www.tcpdump.org/
[4] https://www.tcpdump.org/manpages/pcap_datalink.3pcap.html
[5] https://github.com/nmap/nmap/blob/6a42ef47c08c7a450cefb543fe028bcf991f00b6/tcpip.cc#L1552

between the **20** and **40** bytes. The another field that is using from the **IPv4** Header is *IP_TOTAL_LENGTH*, which represents the length of the data that are carried by this header. Subsequently, according to value in the *IP_PROTOCOL* field is calling the relevant method to processing of the next layer.

The processing of the *IPv6* Header is little more complex. Since, except for the basic type of **IPv6** Header may contains also the *Extension IPv6 Headers.* Mainly for these facts are from the **IPv6** Header derived the *IP6_NEXT_HDR* and *IPv6_HDR_LENGTH* fields. The processing of the individual types of *IPv6 Extension Headers* is based on the reading these two fields from the relevant structures of extension headers. Until the *UDP* or *TCP* protocols appear in the field, extension headers will be skipped with using the obtained length. The format of the *Extension IPv6 Headers* and its meaning you can see in the project code, respectively in the project code documentation. The supported type of *Extension IPv6 Headers* are:

- *Fragment Header*, *Destination Options*, *Hop-By-Hop Options*, *Routing Header*, *Authentication Header*, *Basic IPv6 Header*, *Mobility Header*

In the case of receiving the unsupported type of *Extension IPv6 Headers* or the unsupported protocol the packet will be ignored. At successful processing to *UDP* or *TCP* protocol will be stored the required values, as well as *IPv4* Header and call the relevant method for processing of the next layer.

5.2.3 Transport Layer

Transport layer represents the last part before the payload that are carried by the relevant packet. In this layer are acceptable two most used protocols, *UDP* and *TCP* protocols. The processing of the *UDP* protocol is very simple and consist the obtaining of *DNS* payload, that is locating after the header of this protocol. The structure of *DNS* header you can see in the project code documentation.

The processing of the *TCP* packets is little more complex, since the *TCP* communication can be assembling and therefore is processing of all *TCP* packets as follows. The each packet that will be caught be packet capture handler will be stored to the vector of *TCP Packets.* The reason of its storing is, that the packet have not carry the whole *DNS Payload.*

Subsequently, in case of the request to send a statistics to the syslog server or to report statistics on the standard output will be the vector with *TCP* packets processed. Packets are picked from this vector for individual processing. An important indicator of whether the packet decomposing or not is the field at the begin *DNS Payload* about the length two bytes. This length indicates the length of whole *DNS Payload* under any circumstances - *DNS_LENGTH.* This value is compared with the *TCP Segment Length* and on the basis of that is decisive whether the packet is assembling or not. *TCP_SEGMENT_LENGTH* is computing as difference of *NETWORK_PAYLOAD_LENGTH* and *TCP_HDR_LENGTH.*

In the case when the packet is not assembled will be immediately store to the auxiliary vector of *Reassembled TCP Packets.* Othwerwise, it will proceed in searching of the needed packets to reassembling of whole *DNS Payload.* The next packet from the relevant *TCP Flow* will searching on the basis of *TCP_SEQ_NUMBER.*

After the obtaining of the *TCP_SEQ_NUM* from the currently processing packet will be compared with the *NEXT_SEQ_NUM*, so whether the packet belongs to the currently processed *TCP Sequence.* In the case of the match of *SEQ_NUM* will be packet, respectively its **DNS Payload** attached to the currently processed *TCP Sequence* and will be computing the new *NEXT_SEQ_NUM*. The *TCP Sequence* is complete when the length of the whole *DNS Payload* will equal to *DNS_LENGTH*, then will be the whole Sequence store to the vector of *Reassembled TCP packets* [5].

After the processing of whole *TCP vector* will be the new vector with reassembled packet send to processing of *DNS Payload* and the processed packet will be removed from the *TCP Vector*. The packets that were not processed stay in the *TCP Vector* and will waiting to own *TCP Sequence*.

5.3 Sending Statistics to Syslog Server

Syslog is a way for network devices to send event messages to a logging server - usually known a *Syslog Server*. The **Syslog** protocol is supported by a wide range of devices and can be used to log different types of events. For example, a router might send messages about users logging on to console sessions, while a web-server might log access-denied events[6].

In our case, the *Syslog* server is given by the *-s* argument and can be entered by *IPv4 or IPv6 addresses* or by its *hostname*. The *Syslog* messages have format defined in RFC5424[7]. The example of message, with the one statistics field can see as follows:

- **PRI** - contains the three, four or five characters and will be bound angle brackets as the first and last characters. The number contained within these angle brackets is known as the **PRIVAL** and represents both the *Facility* and *Severity*. The **PRIVAL** is calculated by first multiplying the *Facility* by 8 and then adding the numerical value of the *Severity*.
  - **<134>** = **16** (*Facility* = *local0*) *** 8 + 6** (*Severity* = *Informational*)
- **VERSION** - denotes the version of the **syslog** protocol. RFC5424 uses a **VERSION** value of **„1“**.
  - **<134>1**
- **TIMESTAMP** - formalized timestamp derived from RFC3339[8].
  - **<134>1 2003-10-11T22:14:15.003Z**
- **HOSTNAME** - identifies the machine that originally sent the syslog message. This field should contain the *hostname* and the **domain name** of the originator in the specified format. This format is called a **Fully Qualified Domain Name** (FQDN). The all possible options for content in this field are: *FQDN*, *Static IP address. hostname*, *Dynamic IP address*, *the NILVALUE*
  - **<134>1 2003-10-11T22:14:15.003Z 1.1.1.1**
- **APPNAME** - identify the device or application that originated the message. It is a string without further semantics. It is intended for filtering messages on a relay or collector.
  - **<134>1 2003-10-11T22:14:15.003Z 1.1.1.1 dns-export**

[6] https://www.networkmanagementsoftware.com/what-is-syslog/
[7] https://tools.ietf.org/html/rfc5424
[8] https://tools.ietf.org/html/rfc3339

- **PROCID** - identify the process that originated the message. In our application is using built-in function `getppdid()`[9] to inserting this value to syslog message. The next two field are not used, concretely they are *MSGID* and *STRUCTURED-DATA* fields.
  - **<134>1 2003-10-11T22:14:15.003Z 1.1.1.1 dns-export 1 - -**

The last part of *syslog* message is supplemented by logging **MESSAGE**. The optimal length of this message is **1024B**. At creating of this part in our application are attached the individual items of statistics, while the total length of the message does not exceed this size. Subsequently, the created message is sending to the address that was obtained according to given syslog identification.

Parameter *-t* determines the interval for sending the statistics at *online* mode of application. When there is a time when the statistics are to be sending, in the *alarm* handler will happen the fork() of the main process. The new child process will sending the statistics to syslog server and then dies. Better said, finds itself in a state when was terminated but not reaped by its parent (*defunct (zombie) process*). The main process after the fork() set the new alarm for the next round of sending and will continues at sniffing at the interface.

[9] https://linux.die.net/man/2/getppid

Bibliography

[1] Domain names - implementation and specification. RFC 1035. November 1987. doi:10.17487/RFC1035.
Retrieved from: https://rfc-editor.org/rfc/rfc1035.txt

[2] Arends, R.; Sisson, G.; Blacka, D.; et al.: DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155. March 2008. doi:10.17487/RFC5155.
Retrieved from: https://rfc-editor.org/rfc/rfc5155.txt

[3] Matoušek, P.: *Sít'ové aplikace a jejich architektura*. VUTIUM. 2014. ISBN 9788021437661.

[4] Rose, S.; Larson, M.; Massey, D.; et al.: Resource Records for the DNS Security Extensions. RFC 4034. March 2005. doi:10.17487/RFC4034.
Retrieved from: https://rfc-editor.org/rfc/rfc4034.txt

[5] Singh, A.; Subhlok, J.: Reconstruction of application layer message sequences by network monitoring. In *IASTED International Conference on Communications and Computer Networks*, vol. 151. 2002.

[6] Sinha, S.: Removing DNS as a Single Point of Failure. online. 2017-03-17.
Retrieved from:
https://blog.shrey.io/removing-dns-as-a-single-point-of-failure/