DNS Server

Network Applications and Network Administration

November 20, 2016

Author: Marek Bielik (xbieli05@stud.fit.vutbr.cz)

Faculty of Information Technology
Brno University of Technology

Contents

1 Overview

RoughDNS is a recursive, concurrent DNS server, which supports various resource record types and runs over UDP on Unix-like systems. It is written in C++ according to RFC 1035 [2] and it also supports multiple zone files (utilizing an open source library [1]).

2 Features

2.1 Usage

`./roughDNS [-m <ip_addr>] [-h] [-p <port#>] [<zonefile1>] [<zonefileX>]`

- `-m [--mitm] <ip_address>`

  Responds to every A and MX query with the given IP address. This parameter simulates the man in the middle attack. Other records are served in the standard way.

- `-h [--help]`

  Prints out a simple user manual.

- `-p [--port] <port number>`

  The server will bind its socket with the given port number. If not specified, the standard port number will be used - 53.

The last optional parameters are the paths to zone files. The resource records specified in the zone files must be unique.

2.2 Supported resource record types

- A
- AAAA
- MX
- CNAME
- SOA
- NS
- PTR
- TXT

2.3 Description of the supported resource records

The resource records which the server is authoritative for can be loaded from a zone file. The server also supports multiple zone files. In this case, the records in every zone must be unique. The user can specify arbitrary number of the zone files to be loaded by typing the paths as the last parameters on the command line while launching the server.

When the server doesn't find any record for the requested domain name, it resolves the client's query recursively and sends back a non-authoritative answer. The server also supports the Authority and Additional sections of a DNS message.

When the server receives a query or sends a reponse, it also prints the information to the standard output. Examples are shown in the section 4. If the answer is empty, the server won't print anything but the question.

If the client is looking for a record which happens to be 'behind' a CNAME record in the zone file, the server will look for every occurence of the actual canonical name and send both the desired record as well as the CNAME record. Recursive CNAME records are also supported. Example 4.2 shows this case.

3 Implementation

The server is written in C++ and it makes use of an open source library [1] in order to parse the zone files and to resolve the non-authoritative requests. It is supposed to run on Unix-like systems providing the library has been installed. The server also utilizes Berkeley sockets for UDP communication with clients which is done manually without utilizing any third-party libraries. It supports only the IN class of resource records and only one question per message.

In order to keep the length of the message under the limited size (512B) [2] section 2.3.3, the server supports the compression scheme of domain names introduced in RFC 1035 [2] section 4.1.4. The compression eliminates the repetition of domain names in a message. In the case that the length of a message exceeds the maximum length, the server will truncate the message and set the truncation bit.

Since it is a concurrent server, it forks a new process for every new incoming message and processes it separately.

The server can be terminated by receiving the SIGINT signal. It has been tested in Valgrind for potential memory leaks and none were present (even in the forked processes).

4 Examples

4.1 Example1:

```
q: foo: type SOA, class IN
r: foo: type SOA, class IN, mname a.root-servers.net.
q: foo: type A, class IN
r: foo: type A, class IN, addr 85.214.123.64
r: foo: type A, class IN, addr 85.214.123.63
q: foo: type MX, class IN
r: foo: type MX, class IN, preference 50, mx mx1.foo.
q: www.foo: type AAAA, class IN
r: www.foo: type AAAA, class IN, addr 2001:67c:1220:809::93e5:917
q: 85.214.123.64.in-addr.arpa: type PTR, class IN
r: 85.214.123.64.in-addr.arpa: type PTR, class IN, www.foo.foo.
q: foo: type NS, class IN
r: foo: type NS, class IN, ns l.root-servers.net.
q: txt.foo: type TXT, class IN
r: txt.foo: type TXT, class IN, txt "foobar"
```

In this example we can see the standard output for the supported resource records. There is a SOA, A, MX, AAAA, PTR, NS and a TXT record listed respectively. `q` stands for a question and `r` stands for a response.

4.2 Example2:

```
q: www.foo.ex: type A, class IN
r: www.foo.ex: type CNAME, class IN, cname bar.foo.ex
r: bar.foo.ex: type CNAME, class IN, cname foo.ex
r: foo.ex: type A, class IN, addr 85.214.123.64
r: foo.ex: type A, class IN, addr 85.214.123.63
```

The client is asking for an A record of the `www.foo.ex` domain name. We can see that there is one CNAME record for this name with the canonical name `bar.foo.ex` and that there's also another CNAME record for this name as well. After these two CNAME records, there are two A records which satisfy the original request.

References

[1] NLnetLabs ldns. https://www.nlnetlabs.nl/projects/ldns/, accessed: 2016-11-20.

[2] Mockapetris, P.: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. RFC 1035.
URL http://tools.ietf.org/html/rfc1035