BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DNS TUNELLING

PRENOS SÚBOROV S POUŽITÍM DNS

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **SAMUEL DOBROŇ**
AUTOR PRÁCE

BRNO 2022

Contents

Chapter 1

Traffic tunneling using DNS

Tunneling using DNS is just yet another method used to transfer data over networks that somehow block outcomming traffic. DNS traffic is usually allowed to pass since client needs to resolve host name that could be replaced by captive portal. This project implements tunneling using bastion hosts (see [3]).
The gist of whole tunneling over DNS is to encode data to the DNS packets whose looks like regular traffic. There are not many options where data could be encoded to, user's data could be freely put in `QNAME` part of DNS query only. However this method requires encoding binary data to its' text represenatation using `base32` encoding if queries type `A/AAAA` are used which leads to big overhead (see [1] for host name limitations). Another parts of DNS header could be also used but non-standard header could be problem for firewall in the network. For example last 13bits of `OPCODE` field, those are reserved for future use and should be safe to use for data. See `RFC1035`[2].

Chapter 2

Implementation

Project is logically separated into multiple directories:

- `receiver` - receiver source code
- `sender` - sender source code
- `common` - common parts such as `base64` implementation

All the source codes are written in C11. If you use provided `Makefile`, it takes care of everything, if not, used `CFLAGS` were `-O2 -g -std=c11 -pedantic -Wall -Wextra`.

2.0.1 Communication protocol client - server

Communication between client and server is designed to be as sneaky as possible. That's the reason of using randomly generated `ID`.

Handshake

As a handshake in this project is meant process of setting up connection from client (sender) to server (receiver). It is done in 3 steps:

1. Hello UDP packet
    - Sender sends valid DNS query packet using UDP with `QTYPE` set to `A` and `QNAME` set to `BASE_HOST` domain.
2. Hello UDP packet response
    - Receiver replies with valid DNS response packet using DNS with `TC` flag set to `1` which based on documentation means, client should use TCP instead.
3. Opening TCP connection
4. Filename packet
    - Sender sends valid DNS query packet with filename encoded in `QNAME` with `BASE_HOST` suffix.
5. Server checks the file
    - Server tries to open file, if it fails, it closes the connection.
6. Server is ready to receive file

File transfer

After successful handshake, server waits for packets with data.
Data are encoded into `base64` chunks with `BASE_HOST` suffix. String that results from this operation is then copied into `QNAME` and DNS query is send to the server. Client is then waiting for server to decode, de-chunkize data and write it to the file if everything was successful, which aside of that means data were transported correctly, server takes received packet, changes both `QR`[1] and `TC`[2] flag to `1` and sends it back to the client. Client treats this packet as a acknowledgement and sends next chunk of encoded data. As RFC2181 [1] says, only host names requires specified format of labels. That means, labels of `A` and `AAAA` records could contain alphanumeric characters only. For that reason I have chosen `CNAME` type instead of `A` or `AAAA` for data transfer as my implementation uses `base64` (see [?] for details).

2.0.2 Data encoding

Data are encoded using `base64` encoding algorithm. File chunks are encoded to `a-zA-Z0-9+-` characters and `=` as padding character. Implementation of `base64` is taken from https://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.c (under BSD license).
After file chunks encoding, data are labeled into 63 characters long labels divided by `.` ending with `BASE_HOST`. This kind of encoding respects `RFC1035`[2].

2.0.3 Extensions

IPv6

Application does not make difference between `IPv4` and `IPv6` addresses/packets. Networking is implemented to support dual-stack, `IPv4` addresses are mapped to `IPv6` addresses[?]. That results into full support of `IPv6`.

DNS resolution

Receiver (or server) does not implement file transfer only. For queries that are not sent with `BASE_HOST` suffix regular DNS resolution is performed.
In that case, receiver acts as a proxy. It takes DNS query and forwards it to the DNS server configured in `common/dns.h` file, by default, `1.1.1.1` respectively `2606:4700:4700::1111` Cloudflare DNS is used. As receiver receives response, it forwards whole response to the client (see [?] for testing with `dig`).

[1] Query/Reply - Which means the packet is reply.
[2] TrunCated

Chapter 3

Testing

3.0.1 Functional testing

Receiver has been tested using `dig`[1]:

```
$ dig @127.0.0.1 vutbr.cz
; <<>> DiG 9.10.6 <<>> vutbr.cz
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10491
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;vutbr.cz.   IN A

;; ANSWER SECTION:
vutbr.cz.  191 IN A 147.229.2.90

;; AUTHORITY SECTION:
vutbr.cz.  1858 IN NS rhino.cis.vutbr.cz.
vutbr.cz.  1858 IN NS pipit.cis.vutbr.cz.

;; ADDITIONAL SECTION:
pipit.cis.vutbr.cz. 1 IN A 77.93.219.110
pipit.cis.vutbr.cz. 1 IN AAAA 2a01:430:120::4d5d:db6e
rhino.cis.vutbr.cz. 65 IN A 147.229.3.10
rhino.cis.vutbr.cz. 65 IN AAAA 2001:67c:1220:e000::93e5:30a

;; Query time: 54 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Nov 10 23:25:16 CET 2022
;; MSG SIZE rcvd: 185
$
```

[1] https://linux.die.net/man/1/dig

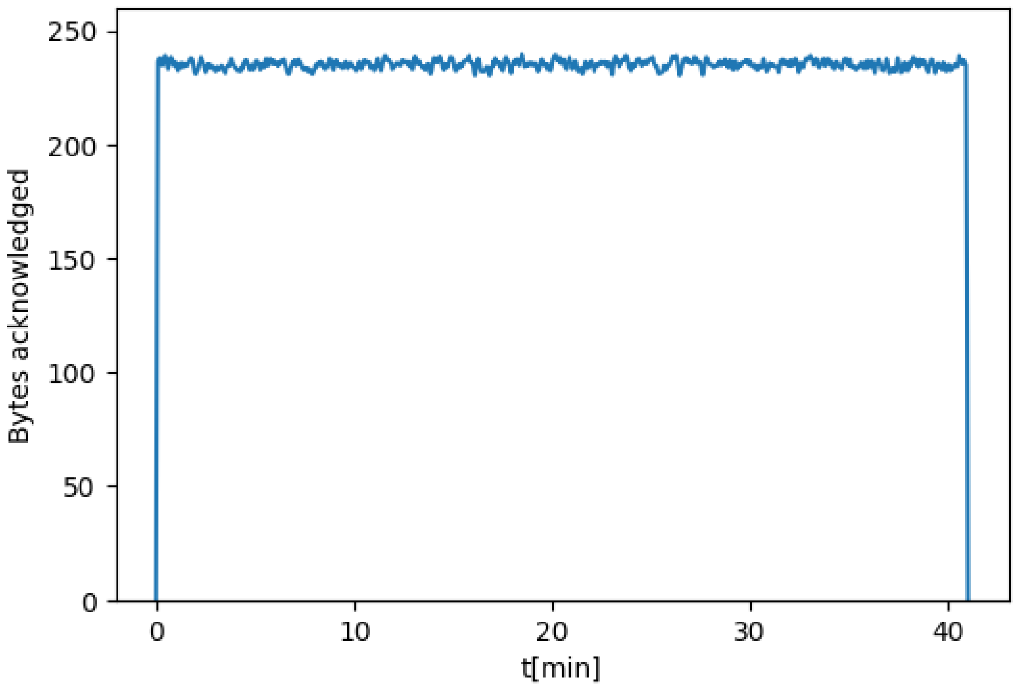
And also with `diff`[2] tool to check whether sent file is same on both sides:

sender	receiver
	`$ ./dns_receiver dobron.sk tmp`

`$ ./dns_sender -u ::1 dobron.sk file bin`
`...`
`[CMPL] bin of 104254B`
`$`
	`...`
	`...`
	`[CMPL] tmp/file of 69240B^C`
	`$ diff bin tmp/file`
	`$ echo $?`
	`0`

3.0.2 Performance testing

Throughput of tunnel could be measured setting macro `MEASURE` to `1` in `sender/sender.h` file and recompile sender. With this macro, parts of code that prints number of sent (and acknowledged) bytes (excluding `BASE_HOST`) with current time in microseconds are compiled.
Throughput was tested with `dobron.sk` as a `BASE_HOST` and `10Mb` random generated file:



[2]https://man7.org/linux/man-pages/man1/diff.1.html

Bibliography

[1] Elz, R. and Bush, R. *Clarifications to the DNS Specification* [Internet Requests for Comments]. RFC 2181. RFC Editor, July 1997. 13 p. Available at: https://www.rfc-editor.org/rfc/rfc2181#section-11.

[2] Mockapetris, P. *Domain names - implementation and specification* [Internet Requests for Comments]. RFC 1035. RFC Editor, November 1987. Available at: https://www.rfc-editor.org/info/rfc103.

[3] Pearson, o. *DNS Tunnel - through bastion hosts*. Seclists, April 1998. Available at: https://seclists.org/bugtraq/1998/Apr/79.