

# DNS tunneling

Rastislav Budinsky(xbudin05)

October 8, 2022

## Abstract

This paper is written in first person plural but this whole assignment was implemented and written only by the author of this paper (yes me, xbudin05). I just like to write it this way, hopefully this will not be held against me.

## 1 Introduction

This project creates a tunnel between sender and receiver imitating DNS communication. This way a communication to the outer observer might appear as almost normal and valid DNS communication. This might not be held true against every packet transferred. This will be discussed later in [Drawbacks](#).

**Note:** This project might not work against actual filtering of normal communication, please keep this in mind when using it.

## 2 Settings

When only small packet is transferred, meaning the whole size will not surpass 512B defined by the [\[rfc\]](#) the program should work as expected from the sender and receiver side.

However, when data transferred is larger, then TCP connection is used. There is no problem with TCP connection per se but rather with how it is utilized in this project. [\[rfc\]](#) defines also standard for TCP DNS queries. However because of flawed design in the early stage we do not consider it to be compliant. There are 2 particular problems, which should not be hard to fix in case they are critical for proper functionality.

Problems are discussed later with how [DNS header](#) is created and utilized and how packets are sent using [TCP's buffer](#).

**Important:** As we did not properly understand the assignment we do not know we should work with subdomains so the only behavior implemented that there is, we check if the domain name from the query and domain name provided on startup of the receiver are the same, otherwise we do not accept this communication.

## 3 Drawbacks

- We use number of questions to signalize if TCP communication is needed. 1 (default and normal) states only UDP, 2 means switching to TCP is required. To change this behavior a little bit of change would be needed, where we would append behind the filename one byte of data, signaling this property.
- On sender side we read almost TCP limit (64000B instead of 64kB) and once filled we send it, let TCP to parse it into smaller packets. We have not tested if this form would get filtered out, as only the first packet contains almost correct DNS communication, except [DNS header flaw](#). All upcoming packets then contain just chunked payload. Changing this behavior in code should be quite easy. As we would need to read by smaller chunks and update the DNS's resource payload. On Receiver we would by default skip the DNS header, which is now only special case for first received packet. However this would lower our effective bandwidth so we have stuck with the current solution, as we can not prove it not working.

- When doing testing on `eva.fit.vutbr.cz` we found a problem with slow processing as receiver so we have decided to add sleep after every TCP buffer was sent from the sender, making it slower down the communication a bit in this case. However on regular laptop during tests this was not spotted.
- Due to expected reply from the server, which is implemented in asynchronous way, we have to give small time for the forked process to maybe receive reply from the receiver to log the action.
- In case we have to resolve the domain and use DNS server, if the server returns answer as NS and not an actual IPv4 address we only log this action and do not repeat our query. We would never be able to test this feature properly so we have decided to only log this event and let user deal with it.

## 4 Extensions

In this section we describe all extensions added to the project, which might not be explicitly stated in the assignment or are a real extension to the project.

### 4.1 Ceasar cipher

This behavior is implemented by default and can be disabled using flags `-d`. This simple symmetric encryption of data for safety reasons.

### 4.2 Backwards communication

In the sender program we have decided to fork another process which is responsible for logging replies from the receiver. This causes small [slowdown](#) on shutdown. However we consider asynchronous communication to be more efficient compared to waiting for replies and then continuing. Slowdown is there just to wait for the final message to confirm correct finish.

Even-though we do not expect anybody to update our code base, we have refactored the code for a few times to make it more readable and updatable in case of further updates or trying to fix some of [Drawbacks](#).

## 5 Testing

For testing purposes we have provided 2 files called `short.log` and `long.log` which contain 2 different messages each using different protocol for transfer. Expected behavior is to setup receiver and then use sender to send one of the files. You can use tool `diff` and both files should be equal.

## 6 Special thanks

Goes to the author of DNS example [\[fff\]](#), which we heavily used in our project.

## References

[\[fff\]](#) fffaraz. Github. <https://gist.github.com/fffaraz/9d9170b57791c28ccda9255b48315168>.

[\[rfc\]](#) Rfc 1035. <https://datatracker.ietf.org/doc/html/rfc1035>.