DNS tunneling

Rastislav Budinsky(xbudin05)

October 14, 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 transfered, 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.Problems are discussed later with how DNS header is created and utilized and how packets are sent using TCP's buffer.

We would like to note, that data is not encoded in the domain name as it might be expected but rather it is in the payload in resource data section.

Important: As we did not properly understand the assignment we do not know if 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 query type to signalize if TCP communication is needed. We use IPv4 query type (A - default and normal) states only UDP and Name server query type (NS) 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, signalizing this property.
- For transfering data we use resource data for payload which might get easily filtered by little bit advanced anti DNS tunneling service. However the packets are still valid DNS packets, even if sometimes they are sent in batches. Meaning by default we use total TCP payload to 1024B (so we can do 64 of DNS queries per one TCP connection) but it might get buffer and sent for example by 4. This shows as TCP DNS communication with 4 queries.
- 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.
- Receiver is waiting for the whole TCP packet so it can parse it properly or until the TCP packet states end of communication. If any packet get corrupted or wrong lentgth is sent, receiver might get stuck in a loop waiting for whole packet or end of communication.

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.

Testing was done with multiple setups:

- from one laptop to another laptop with VM. Using only local communication, as second laptop was connected via hotspot. Not using DNS port
- from one laptop to eva.fit.vutbr.cz server . Not using DNS port
- on loopback using DNS port.

Note: If you would like to change port navigate to `/helpers/dnsHeader.h` and change macro PORT

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.