Vysoké učení technické v Brně
Fakulta informačních technologií

ISA – project

DNS Tunneling

November 12, 2022

Patrik Korytár (xkoryt04)

Contents

1 Theory and motivation

1.1 DNS tunneling motivation

Oftentimes, firewalls protect or restrict communication across network so only data of certain application protocols can be received in / sent outside of network. Application protocols typically have certain standard port assigned. That means, firewalls check for port in TCP/UDP header and only let pass packets of some protocols.

DNS is one of those protocols which are essential in the network (in order for hostnames to be able to be translated into IP addresses) and therefore firewalls do not block it.

What if we want to receive data from outer network or send data to outer network, but firewall restricts necessary protocols? We can use, for example, DNS tunnneling.

1.2 DNS tunneling

DNS tunneling provides a way to send various data by the means of standard DNS communication - by using DNS queries and responses. All data are transformed into DNS messages in such a way that they form valid DNS protocol messages and yet their meaning is different from standard one.

One way to achieve this (and as is implemented in this very program) is as follows. We need to register a DNS domain (named same as base hostname used in the program) and establish our own authoritative server for the domain[1]. On the client side, data will be transformed into hostname. Data need to be divided into blocks and each block encoded using for example base32 so valid hostname can be formed. Then encoded data block needs to be divided into labels with base hostname appended to the end. The string now forms valid DNS hostname and will be included in DNS query that client can send. When DNS query is sent, DNS resolution is used. So (if we registered our domain) after proper DNS resolution, our query should arrive to the server authoritative to the base hostname domain. This way client inside one network protected by firewall can send data encoded inside DNS to outer network. Possibly also vice versa.

Both client and server need to know meaning of the data in advance, so server can respond accordingly. Server then forms valid DNS response and sends it back (response arrives using this DNS resolution chain). For encoding of response, DNS flags or response IP can be used, for example.

This reply-response cycle repeats until all data are exchanged.

1.3 DNS resolution

Translation of DNS hostname to IP address proceeds in multiple steps. Firstly, query is sent to conigured (typically local) DNS server. Then queries are sent to root DNS server, top level domain server, and progressively so until server authoritative to the last subdomain is reached [1]. This works into our benefit. When DNS query will arrive to our server authoritative to our base hostname domain, it will be the last server receiving our query (since our server won't redirect to other DNS server). Our server will act and respond respectively, as we want to.

[1]My implementation is simplified - it's possible to specify IP address of the server to which data will be sent and no domain registration is needed.

2 About the program

2.1 How to use the program

Description on how to run the program, list of parameters and their usage is described in the README.md file.

2.2 Technologies used

Implementation uses C11 language and its standard library. Program *make* is used to automate the compilation.

2.3 Libraries used

Program uses standard linux socket and networking libraries (sys/socket.h, arpa/inet.h, netinet/in.h).

2.4 Code documentation

Code is documented using Doxygen comments. Please refer to the code and commented functions to get more information about the implementation. High-level description is included below.

2.5 Protol stack

Protocol stack consists of IPv4, TCP (not UDP) and DNS.

3 Code structure

Code files are included in the *src* directory.

Client code is divided into following modules:

- dns_question – DNS query construction
- dns_sender – creation of client and connection to the server

Server code is divided into following modules:

- dns_response – DNS query parsing and response construction
- dns_receiver – creation of server and connection to the client

Common modules:

- dns – basic DNS structures
- utils/args – argument parsing
- utils/generic – generic functionality

Note

Modules utils/base32, dns_sender_events and dns_receiver_events are not implemented by me. base32 module's author and license is mentioned in the code. sender modules were provided in the assignment.

4 Encoding

4.1 Encoding of file data

Client encodes tunneled data (be it filename or file contents) using base32 algorithm, so these data can be used to form valid DNS hostname. Encoded data are splitted into multiple labels, while label is no longer than 63 bytes. Encoded data are appended with base hostname. All labels are preceded by length byte of respective label, whole hostname ending with NULL byte. Final string is included in DNS query question QNAME field (and copied into DNS reply).

For example, filename test.txt with base hostname random.com is encoded and sent as hostname:

```
{13}ORSXG5BOOR4HI{6}random{3}com{0}
```

Where `{}` denotes length byte. In human readable format: `ORSXG5BOOR4HI.random.com`.

4.2 Encoding of response code

Server denotes response code in DNS reply resource record RDATA field (A type format) - as four byte long IP address. Depending on what type of error occured - or possibly no error occured - respective IP address is included in reply. Therefore, client also needs to have these IP addresses hardcoded in the code, so response code can be understood.

For example, server could not open file, so IP address 2.2.2.2 is sent in response. Client recieves this response, reads it and since knows that 2.2.2.2 IP address means file opening error, ends the connection.

4.3 DNS reply RCODE

Server does not use RCODE but - as mentioned above - RDATA field. Nonetheless, client still checks whether reply RCODE is equal to zero - there is possibility client will connect to other DNS server, not implementing this DNS tunneling protocol.

5 Communication protocol

Client-server protocol works as follows.

5.1 Protocol steps

TCP connection establishment

Firstly, client needs to connect to the server IP address and port. Default DNS port is used.

Client sends filename

Client sends DNS query containing encoded filename (see section 4.1).

Server response to the filename

Server checks if base hostname of the hostname is correct and tries to create file with requested filename (under destination dirpath as specified by program argument). Server then responds with respective IP address depending on state of success of file creation (see section 4.2).

If error occured, connection ends.

Client sends chunk of file data

Client reads chunk of data from local file, encodes it using base32 algorithm, divides it into multiple labels, transforms into valid DNS hostname appended with base hostname and sends DNS query with hostname to the server.

This and next step repeat until whole file was sent.

Server response to the file data

Server checks if base hostname of the hostname is correct, merges DNS labels together (excluding base hostname), decodes data and writes them into the file. Once again, respective IP address is included in DNS reply.

If error occured, connection ends.

End of connection

Once client has sent whole file data, connection is ended. Server then closes file he was writing into.

6 Testing

For the purpose of testing, I tested the client-server stack on loopback, local network and across Internet. Sometime, though, I needed to change port applications use.

I sent file from client to server and checked whether file with specified filename and in specified destination directory was created. I compared the file size and file contents - either automatically (`diff`) - or just by looking at result. I tried both text files and binary files (images).

I always needed to use -u parameter while running DNS sender - I do not own server authoritative to some domain so I could use normal DNS resolution.

6.1 Testing on loopback

I started receiver in one console. Then I run sender in order to send some file to the receiver with DNS IP being equal to loopback. Default DNS port could be used here.

```
sudo ./dns_receiver random.com test/
...
./dns_sender -u 127.0.0.1 random.com test.txt test.txt
```

6.2 Testing on local network

On one PC, I started receiver. On the other PC, I run sender sending file to the receiver's local IP address. Default DNS port could be used here.

```
sudo ./dns_receiver random.com test/
...
./dns_sender -u 10.0.0.10 random.com test.txt test.txt
```

6.3 Testing on the Internet

On Merlin FIT VUT server, I started receiver. On my PC, I run sender sending file to the receiver's remote IP address. Default DNS port could not be used here, since I do not posses administration rights on the said server.

```
./dns_receiver random.com test/
...
./dns_sender -u 147.229.176.19 random.com test.txt test.txt
```

References

[1] Wikipedia: Domain Name System. [online]. Last modified: 10/11/2022 [Accessed: 12/11/2022]. URL `https://en.wikipedia.org/wiki/Domain_Name_System`