INFO0010 Introduction to Computer Networking
Second part of the assignment

Maxime Aerts, Guy Leduc
University of Liège

Academic year 2022 - 2023

Contents

1 DNS Tunneling

1.1 DNS

DNS (Domain Name System) is a service that converts hostnames to IP addresses among others. It is an application layer protocol that allows users and servers to exchange messages. DNS provides another way to identify hosts than their IP address, as IP addresses can change over time and are difficult to remember.

Nowadays, DNS is deployed everywhere and many protocols rely on it for their own operation, making it one of the central pillars of the Internet.

1.2 Tunneling, as intended

Tunneling is a method for transporting data across a network using protocols that are not supported by that network. In the network layer, tunneling works by encapsulating IP packets (header + payload) inside (i.e., as payload of) other IP packets. This can

be used to carry IPv6 packets in a network that only supports IPv4 for example. Thus, tunneling is nothing else than an extra level of encapsulation. Therefore tunneling can in principle be used to encapsulate any protocol (of any layer) into any other protocol.

1.3 Tunneling, not as intended

Tunneling can also be used to "sneak through" a firewall, using a protocol that the firewall would normally block, encapsulated inside a protocol that the firewall does not block, such as DNS, ICMP, or HTTP. As this usage is most of the time not expected, the client and the server are often specifically configured to understand the encapsulation in place and use the encapsulated protocol afterwards.

1.4 Tunneling, in the project

For this project, you are going to implement a simple pair of client-server able to perform DNS tunneling. You already completed a client prototype in the project first part, now in the second part, you will implement a DNS server.

To use DNS tunneling, the DNS client forges a specific DNS query with a payload (here the question) encoded according to an encapsulated protocol agreed between the DNS client and the DNS server. The (adapted) DNS server is designed to understand this unusual query, react accordingly, and reply to the DNS client. An intermediate firewall will just see normal DNS exchanges and will likely allow them, thus allowing any kind of forbidden data exchanges.

2 DNS server

Your DNS server will listen to TCP connections on port 53 (or any other port, if you prefer). You can choose to stay single threaded (like the DNS client) or to implement multi-threading for handling multiple connections at a time. Submitting a functional multi-threaded server will award you bonus points. Once you got the connection socket in your main thread or in a new thread:

1. Wait for data until data arrives, connection is closed by the client, or more than 5 seconds have passed without activity (timeout).
2. Try to read as many bytes as specified in the DNS TCP header (first 2 bytes). If there were not enough bytes sent, wait for 5 seconds without activity and close the connection.
3. Consider the read bytes as a DNS query, parse it and check if all the variable length indicators are fully respected, if not, close the connection.
4. Parse the DNS query, at least the fields you need for later steps. If there is a format error, reply with a valid DNS response without answer and with response code (RCODE) set to "Format Error".

5. Check that the DNS query contains only one question of type TXT with a name following this pattern: `<tunneled data encoded in base32>.<owned domain name>`. If this requirement is not respected, reply with a valid DNS response without answer and with response code (RCODE) set to "Refused".

6. Decode the "tunneled data" from base32 into a valid URL (base32 padding will be omitted, you need to handle this case). If the URL is not valid, reply with a valid DNS response without answer and with response code (RCODE) set to "Name Error".

7. Perform an HTTP GET request (use a library such as `java.net.HttpURLConnection`) sent to the decoded URL. You should receive a valid HTTP response in text characters. Otherwise, reply with a valid DNS response without answer and with response code (RCODE) set to "Name Error".

8. Encode the HTTP response content in base64 to ensure that all transmitted characters are in ASCII format.

9. Verify that the encoded HTTP response in addition to the rest of the packet does not exceed $2^{16} - 1$ bytes. You can choose to compute the exact available size for encoded HTTP response or simply define an arbitrary maximum of 60000 bytes. If the encoded HTTP response is larger than this threshold, truncate it by taking only the first 60000 bytes (it works with base64 because 60000 is a multiple of 4).

10. **Optional:** For one bonus point, you can choose to format your encoded HTTP response in `<character-string>s` as described in TXT RDATA definiton in RFC 1035. If you do not use `<character-string>s`, standard DNS client such as `nslookup` will trigger a malformed error, but with DNS tunneling you are supposed to control the client and the server, thus it is not necessarily a problem.

11. Craft and send a valid DNS reply with an answer and the HTTP response content (which can be truncated) encapsulated in TXT RDATA of the payload. If the encoded HTTP response has been truncated, set response code (RCODE) to "Name Error", otherwise set it to 0, i.e. no error.

For a simple URL with not too much content, try `https://example.com/`.

3 Program Input/Output

3.1 Input

Your server can be launched using the following command: `java Server <owned domain name>`, where "owned domain name" is a valid domain name under the responsability of your server. Thus, the server will only successfully reply to queries addressing this domain and its subdomains. For example :

- `java Server tnl.test`

3.2 Output

The output of your program will follow a strict format written to `stdout`. It will print one line for each query received. This format is similar to the question printed in your client output, this is: `Question (CL=<IP of the direct client>, NAME=<domain name to query>, TYPE=<question type>) => <reply code>`, where names between `<...>` must be replace accordingly. For example :

```
Question (CL=192.168.0.27, NAME=nb2hi4dthixs6yjomjss6mjopbwwy.tnl.test, TYPE=TXT) => 0
Question (CL=192.168.0.45, NAME=test.truc.be, TYPE=A) => 5
```

4 Guidelines

4.1 General

- You will implement the programs using Java 1.8, with packages `java.lang`, `java.io`, `java.nio`, `java.net` and `java.util`. You can also use an external package to encode and decode base 32 data, but if possible, use "Help files for project 2" available in project content on eCampus.
- This second part of the project can be completed alone or as a group of 2 people.
- Your server should run continuously until it receives an interruption signal (SIGINT), when you hit Ctrl+C in the console for example.
- You will not manipulate any file on the local file system.
- At the root of the submitted archive, you'll have a file `report.pdf` and a directory `src/` containing all your Java code. Other files or directories can be present, but it should not be necessary.
- You will ensure your main class is named `Server`, located in `src/Server.java` from the root of the archive, and does not contain any `package` instruction.
- You can cluster parts of your program (not `Server`) in packages or directories.
- You will ensure that your program is fully operational (i.e. compilation and execution) on the student's machines in the lab (ms8xx.montefiore.ulg.ac.be).

Submissions that do not observe these guidelines could be ignored during evaluation.

Your commented source code will be delivered no later than 11th of December 2022 (**Hard deadline**) to the Montefiore Submission platform (http://submit.run.montefiore.ulg.ac.be/) as a .zip package.

4.2 Report

Your program will be completed with a report called `report.pdf` (in the zip package) addressing the following points:

Software architecture: How have you broken down the problem to come to the solution? Name the major classes and methods responsible for requests processing.

Length limit: In your program, you have a hard limit on the HTTP response size. Explain why and how would you do to overpass this limit.

DNS Tunneling in practice: In this project, you performed DNS tunneling in only one hop, initiating a direct connection between your controlled client and your controlled server. How could you perform DNS tunneling if your DNS client had to send its queries to an uncontrolled DNS server such as 8.8.8.8 ? What is lacking to your server in order to handle this scenario ?

Limits & Possible Improvements: Describe the limits of your program, especially in terms of robustness, and what you could have done better.

4.3 Evaluation

Criteria used in the evaluation include: program robustness (e.g. handle errors without interruptions), code readability, coding style, and quality of the report.

Some penalties will be applied when the instructions are not strictly followed (e.g. program does not compile on ms800 machines, generates warnings, uses the wrong port number or forbidden libraries, etc.).