Assignment: DNS Server & DNS Client

Objective: In this assignment, you will learn about the Domain Name System (DNS) and how it works. You will also learn how to write a simple DNS server in Python and a test Client to verify your Server code.

Remember, for both labs, we are accessing a DNS server on the localhost.

DNS Lab Part 1

This first part is not meant to be challenging; it helps you get a test client up and running while you work on the DNS Server itself.

Part 1: Introduction to test clients and DNS

A test client is a program that sends requests to a server, and the 'test part' is us verifying that the responses are correct. In the context of a DNS server, a test client sends DNS queries to the server and checks that the responses contain the expected IP addresses. In this case, we are going to query our local DNS server as well as a Public DNS server to see if the responses are the same!

The provided code is an example of a simple test client written in Python using the `dnspython` library. This library provides both high- and low-level access to DNS. The high-level classes perform queries for data of a given name, type, and class, and return an answer set.

The code defines two functions: `query_local_dns_server` and `query_dns_server`. The first function sends a DNS query to a local DNS server running on the same machine as the test client. The second function sends a DNS query to a public DNS server.

Both functions take as input a domain name and return the IP address associated with that domain name according to the respective DNS server. There are a few extra functions there for you to use as you see fit.

Part 2: Writing a Test Client for a DNS Server in Python

In this part of the assignment, you will write your own test client for a DNS server in Python. You can use the provided skeleton code as a starting point, you are to fill in the blanks denoted by '?'. Do not edit outside of the designated blocks.

Here are the steps for writing the test client. Some have already been handled in the skeleton code:

1. Import the necessary modules, including `dns.resolver` from the `dnspython` library.

2. Provide the needed variable values for the local_host (what IP is reserved for local servers on the local machine?), a real external DNS server, and question_type.

3. Define two functions: one for querying your local DNS server and one for querying a public DNS server. Both functions should take as input a domain name and return the IP address associated with that domain name according to the respective DNS server.

4. In each function, create an instance of `dns.resolver.Resolver` and set its `nameservers` attribute to the IP address of the respective DNS server.

5. Use the `resolve` method of the `Resolver` instance to send a DNS query for the given domain name and record type (e.g., `A` for IPv4 addresses).

6. Extract the IP address from the response using the `to_text` method of the first answer record.

7. Return the IP address from each function.

8. Write additional code to call your functions with different domain names and verify that they return the expected IP addresses.

9. Test the results from your local DNS server against your chosen public DNS server, by writing a comparison function that returns a Boolean. See the compare_dns_servers() method.

Use the skeleton code as a starting point, but make sure you understand what each part of it does! Don't modify outside of the designated areas. Once complete, label the program DNSClient.py' and upload it to GradeScope under *'Python Programming: DNS Part 1'*.

DNS Lab Part 2

Part 1: Introduction to DNS (refresher on Kurose/Ross)

The Domain Name System (DNS) is a distributed database that maps human-readable names (such as www.example.com) to IP addresses (such as 192.0.2.1). This makes it easier for users to access websites and other resources on the Internet, as they don't have to remember the numerical IP addresses.

When you type a URL into your web browser, your computer sends a DNS query to a DNS server to resolve the hostname (e.g., www.example.com) into an IP address. The DNS

server looks up the hostname in its database and returns the corresponding IP address to your computer. Your computer then uses this IP address to connect to the webserver hosting the website you want to visit.

Part 2: Writing a Simple DNS Server in Python

In this part of the assignment, you will write a simple DNS server in Python using the `dnspython` library (read up on it!). This library provides both high- and low-level access to DNS. The high-level classes perform queries for data of a given name, type, and class, and return an answer set. The Skeleton Code is a starting point and already completed some of the work for you (link). You will also **exfiltrate** some data by hiding it in a DNS record!

Your job is to fill in the '?' within the skeleton code. Below are the steps we are completing in the code:

1. Import the necessary modules, see skeleton code

2. Define Encryption and Decryption Functions:

   Implement the generate_aes_key, encrypt_with_aes, and decrypt_with_aes functions. These functions are used to generate AES keys, encrypt data, and decrypt data using a provided password and salt.

   We have finished the generate_aes_key method with some presets. It is your job to call the method by passing it the password and a salt to encrypt, and also to decrypt. Use the existing methods, make sure to complete the code.

3. Prepare Encryption Parameters

   The salt is 'Tandon' - make sure to encode as a byte-object!
   The password is your NYU email address registered in Gradescope
   The secret data is: "AlwaysWatching"

   Read more about Fernet here: Fernet (symmetric encryption) — Cryptography

   What we are doing here is storing private information in a DNS type of TXT - what type of variable holds plaintext?

   We are encrypting it so others looking at the data don't know what it says. This is known as a form of exfiltration! We store the record and then externally make a DNS request for it. If we breached a system that has a lot of tools set to check inbound and outbound traffic, it may be hard for us to exfiltrate data, but perhaps they aren't properly checking their DNS Servers? Do not modify (decode or

decrypt) your package being exfiltrated, just make sure it fits the variable type expected by the DNS TXT record.

4. Create a dictionary containing the below DNS records mapping hostnames to IP addresses.

   Hint: 'remember/lookup what a Fully Qualified Domain Name (FQDN) is. This is a good *start* as well (link1, link2). See section 3.1 of RFC 1034 and section 5.1 of RFC 1035.

   You need an absolute name.' We have as an example already filled out example.com for you. You **do not** need to look up any additional record information; only use what is provided below.

   A Records:

   safebank.com -> 192.168.1.102
   google.com -> 192.168.1.103
   legitsite.com -> 192.168.1.104
   yahoo.com -> 192.168.1.105
   nyu.edu -> 192.168.1.106

   For nyu.edu also create the following records:

   TXT -> a string cast version of your encrypted secret data from step 3
   MX -> 10, mxa-00256a01.gslb.pphosted.com.
   AAAA -> 2001:0db8:85a3:0000:0000:8a2e:0373:7312
   NS-> ns1.nyu.edu.

Learn more about IPV6 compression here: RFC 5952: A Recommendation for IPv6 Address Text Representation (rfc-editor.org)

5. Create a UDP socket using `socket.socket` and bind it to the local IP address (???') and port ?? (the standard port for DNS) using `socket.bind`.

6. Receive data from a client using the `socket.recvfrom` method of the socket object.

7. Parse the received data as a DNS message using the `from_wire` method of the `dns.message` module.

8. Create a response message based on the request using the `make_response` method of the `dns.message` module.

9. Get the **first** question from the request and extract its name and type.
10. Check if the name and question type are in your dictionary of DNS records.
11. Depending on the question type you may need to handle additional work.
12. Set the AA (Authoritative Answer) flag manually using bitwise operations.
13. Send the response back to the client using the sendto method of the socket object.
14. Repeat steps 5-13 for each incoming query. Note how the skeleton code uses threading and enables the ability to enter 'q' in order to quit the infinite loop.

Use the skeleton code as a starting point, but make sure you understand what each part of it does! Once complete, label the program DNSServer.py' and upload it to GradeScope under *'Python Programming: DNS Part 2'*.

Deliverables:

Submit both your completed Python scripts for the simple DNS server ('simpleDNS.py') and simple DNS client ('simpleClient.py') on Gradescope under the sections designated. When facing errors, read the instructions carefully to make sure you are handling everything correctly. Read, and research! Do not edit outside of the designated areas. No extra functions are needed, and no function should return in a format different than what has already been specified, it's your responsibility to make it work!

Remember, you should run the client and server code locally and test out your work! To do so, make sure DNSServer.py (server) is running, and in a new window/terminal run DNSClient.py.