ECS 152A: Computer Networks Project 2, Part I

DNS client from scratch

Professor Zubair Shafiq

Krystal Chau, SID: 920918540 Jacob Feenstra, SID: 921423591

November 20, 2023

1 File Submissions

The following files were submitted along with this `proj1` PDF.

1. `main.py`
2. `packQuery.py`
3. `sendingToServer.py`
4. `unpackingResponse.py`
5. `makeHTTPRequest.py`
6. `tmz_payload.html`
7. `gpt_part1.py`

2 DNS packet Creation

For creation of the DNS packet, we referenced the documentation provided in RFC 1035, as well as parts of the course's textbook. Each item below is a 16-bit field in the DNS packet. The order of the list indicates the order in which the fields were created. Sub-fields (such as flags) are further detailed under their respective field.

1. `Identification`: Arbitrary field that gives the packet an identifier. Helps in correlating the DNS response to the DNS query. Encoded as `b'JK'`
2. `Flags`: Field that has various flags specifying query handling, and response status.
   (a) `QR`: Single bit, 0 if a query packet, 1 if a response packet. Set to 0.
   (b) `OPCODE`: 4-bits, specifies kind of query. Set to 0000 for standard query.
   (c) `AA`: Authoritative Answer. 1 in response packet if we have an Answer to our Query. Set to 0
   (d) `TC`: 1-bit Truncation. 1 in response if DNS Response was too big for UDP packet. Set to 0
   (e) `RD`: 1-bit recursion desired. 1 in Query if we wish to use recursive DNS traversal instead of iterative DNS traversal. Set to 0.
   (f) `RA`: 1-bit recursion available. 1 in Response if recursive traversal was available
   (g) `Z`: 3 bits reserved for future use. Must be 000.
   (h) `RCDOE`: 4 bits that indicate if an error was in the response packet. Set to 0000
3. `QDCOUNT`: Specifies number of resource records in question section, for response. All 16 bits are 0.

4. `ANCOUNT`: Specifies number of resource records in answer section, for response. All 16 bits are 0.
5. `NSCOUNT`: Specifies number of resource records in authority section, for response. All 16 bits are 0.
6. `ARCOUNT`: Specifies number of resource records in additional section, for response. All 16 bits are 0.
7. `Question Section`: Much more involved, and of variable bits. Encodes the query.
   (a) `QNAME`: Encodes the question itself, `tmz.com`. Our encoding takes the following form, from alphabet to hexadecimal:
      - 3 tmz 3 com 0
      - \x03\x74\x6D\x7A\x03\x63\x6F\x6D\x00

      Each individual alphabetic character is represented as an octet (8 bits). `tmz` and `com` utilize ASCII.
   (b) `QTYPE`: 16 bits specifying the type of query. Chose type A, which maps with the value 1.
   (c) `QCLASS`: 16 bits specifying the class of the query. Chose IN for Internet, which is 1.

When all is said and done, the final DNS query in hexadecimal is:

```
\x4A\x4B\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03\x74\x6D\x7A\x03\x63\x6F\x6D\x00\x00\x01\x00\x01
```

We send this off to get our DNS Responses!

3 Parsing the Response

Parsing the DNS response involves traversing the DNS server hierarchy. The list below indicates the parsing logic in chronological order.

1. Receive our response from a Root DNS server. One of the servers from this list. Upon receiving the response, we look for resource records (RRs) which store a TLD DNS server IPv4. Our method is as follows for recovering these RRs.
   (a) Iterate through the Header section. Acquire the bit stored at the `AA` flag. Then, acquire the bits stored at the`QDCOUNT`, `ANCOUNT`, `NSCOUNT`, and `ARCOUNT` flags. Convert these four flags to their unsigned integer representation.
   (b) Set our indexer, `offset`, to 12. Since the Header section is 12 bytes long, it will immediately move to the Question section.
   (c) Now, skip over the Question section. Increment `offset` by 1 until a null byte is reached. Then, increment `offset` by 5 to skip the null byte, 2-byte `QTYPE`, and 2-byte `QCLASS`. We have now skipped a single Query in the Question section. Do this `QDCOUNT` times to skip over the entire Question section.
   (d) Check if `AA` is 1. If it is, then we wish to iterate through the Answer section. If not, move to Authority and Additional section.
   (e) We have created an extraction handler which, depending on the logic above, will iterate through one of the three sections indicated. Our Root DNS server has no Answers, so it iterates across the Authority and Additional sections of the response. The logic for both is equivalent, and is detailed below.
   (f) Unpack the current RR, storing it's first 5 values (`RNAME`, `RTYPE`, `RCLASS`, `TTL`, and `RDLENGTH`) as integers.
   (g) Determine beginning and ending points of `RDATA` in the RR, using `RDLENGTH` and current position of `offset`.

(h) If `RTYPE` and `RCLASS` are both 1 (it is an Internet-class RR, and is a Type A, meaning that the RR is returning a host address), extract the `RDATA` of this RR. It is an IP.

(i) Only extract if it is an IPv4. Store the IP in a list.

(j) Increment `offset` the length of the current RR, which is $|12 + \texttt{RDLENGTH}|$.

(k) Repeat steps f-i `NSCOUNT` or `ARCOUNT` times (the number of RRs in the given section). Return a list of IPv4's from this section, corresponding to RRs which qualify in the above algorithm.

With the above steps accomplished, we have acquired a list of IPv4's which identify some TLD DNS servers, which were stored in the root DNS server.

2. Now, all of the same logic recurs for the TLD DNS server. Send query to one of the TLD DNS servers identified above. Receive it's response after sending the constructed query (the very same query), parse it using the algorithm above, and return a list of IPv4's from the resource records of Answer OR Authority/Additional. List of IPv4's identify some authoritative DNS servers.

3. Send query to one of the authoritative DNS servers identified above, acquire it's response. Note this response returns an `AA` of 1, so the Answer section's RRs are parsed (as opposed to Authority and Additional). Return a list of IPv4's which correspond to tmz.com! We have acquired the answer to our query.

From here, create an HTTP request and send it to one of the IPv4's returned from the authoritative DNS server. Then, we take the HTTP response, and write the payload to an HTML file. Since the payload is in of itself HTML, this file can be run in a Web browser! We can see what tmz.com has returned to us. It is an HTTP Status Code 403; Amazon CloudFront, tmz.com's server provider, does not authorize our HTTP request to fetch what is located on tmz.com.

4 Addendum

Please note that the requested RTT's are computed in the Python scripts. RTT to the public DNS resolver is computed in the calls to `unpackingResponse.py`, one for each DNS server (root, TLD, authoritative). RTT to tmz.com is computed in `makeHTTPRequest.py`. `tmz_payload.html` contains the payload returned from our HTTP request to tmz.com. The link to the ChatGPT session is below. Note the corresponding source code is in the file submissions: `gpt_part1.py`.

ChatGPT-3.5 Session Link