DNSTAR

Nidhish Bhimrajka Harshita Gupta Daskh Agarwal

Project Idea

This project is implementing a DNS protocol from scratch that protects the privacy of a DNS client by securing the communications taking place with the DNS recursive resolver. When a client queries a resolver, its IP address and the query are the two things that should be kept private. The main idea behind this project is to not give away both of the information to a single entity (Usually, DNS Resolver).

Instead, we have introduced a relay between the client and the resolver. So, instead of directly communicating with the recursive resolver, the DNS client now communicates with the relay, and the relay, in turn, communicates with the resolver. The relay acts as a middleman between the client and the resolver.

The client and the recursive resolver exchange a symmetric key using public-key cryptography. Using this symmetric key, the client encrypts the DNS packets and sends them to the relay. The relay doesn't know the symmetric key and hence doesn't get to know about the contents of the DNS packet. It simply forwards this encrypted packet to the DNS resolver. Upon receiving the encrypted response from the resolver, it simply forwards it back to the client. The relay learns the client's IP address but not the query, and the resolver learns the query but is unaware of the client whose query it was.

There are three significant attacks on DNS:

- DNS Tunneling attack
- DNS Amplification attack
- DNS Replay attack

Our protocol protects the DNS client from all the attacks mentioned above that the state-of-the-art DNS protocols fail to do.

Project Deliverables

- DNS Resolver
- DNS Client
- DNS Relay

Technologies Used

- C/C++
- Python
- Socket Programming

Implementation Details

- **DNS RESOLVER**

    - We have used C/C++ programming language to code the DNS recursive resolver. The resolver listens on port 53 for incoming DNS queries from multiple relays'. The resolver remains oblivious of the querying client.

    - Encryption - The queries sent/received are encrypted using the AES-128-bit encryption. The AES-128-bit key is

exhchanged using public-key cryptography. We have used the RSA algorithm for this purpose as it was the most widely used state-of-the-art algorithm.

- **DNS CLIENT**

    - Our client has been coded in C/C++ programming language. The client queries the nearest relay for domain name resolution which passes the query to the resolver for an authoritative answer. The DNS packet of the client is encrypted using an AES-128-bit key.

    - This key setup is done initially by the client. The client sends its AES key to the resolver using public-key cryptography. (RSA encryption) For this purpose also, the relay is used, because if the client simply exchanges the key with the DNS resolver directly, then the resolver would know the IP address of the client which we do not want to happen. So, this key exchange is done with the help of the relay to maintain the anonymity of the client.

    - To provide an additional level of integrity to the client's data, the resolver generates a hash of the response using the SHA-256 algorithm, encrypts it with the client's symmetric key, and sends it along with the response. The client verifies the integrity of the response by generating a hash of the response and comparing it with the hash received. This ensures that if the DNS packet gets tampered with during the

transit by the relay or some other potentially malicious actor, it will get detected at the client's end.

- **RELAY**

    - A relay is just an intermediary between the client and the resolver to maintain the anonymity of the querying client. It simply forwards the encrypted DNS queries sent by the client to the DNS resolver and then the encrypted DNS resolution sent by the resolver to the client.

    - The relay has been coded in C/C++ programming language. The code for the relay is very simple. It simply listens for DNS queries on a fixed port. It maintains a mapping of the IP address and the query. The query is forwarded to the nearest DNS resolver on port 53. Upon receiving the resolution, it simply checks for the correct IP address in the map to forward this response.

Technical Details

Primary functions used in our implementation are -

1. **format_query():** This function takes the query from the client and converts it into a DNS packet by appending all the headers and required parameters.

2. **query_resolution():** This function has been used on the client side to make a connection with the DNS resolver to get the query resolved.

3. **get_response():** The function gets the response from the DNS resolver, reads the response and gives the result to the client.

4. **lookup():** This function works to gather authoritative answers to the client's query from the nameserver.

5. **recursive_lookup():** This function recursively calls the lookup function until the query has been resolved.

6. **perform_resolution():** This is the main function of the resolver side that calls all the functions to do the lookup to get the query resolved and sends the resolved answer to the client.

These are the major functions that play a role on the client side and resolver side. Apart from that, we have made helper functions, encryption and decryption functions, and functions that play the role between client and resolver communication, i.e. relay.

How are the three significant attacks prevented in our protocol?

1. *DNS Tunneling attack*

    a. We have added an additional functionality in the DNS resolver to analyse the payload (DNS packet) and filter out

malicious queries before redirecting them to the name server.

b. Here, we use an ML model to accurately predict if a query is maliciously crafted or not. If the model predicts “Correct, " the resolver carries out the name resolution, but if the output is “False, " the resolver rejects/drops that query.

c. The ML model is written in Python and trained using tfx pipeline, where the training dataset is uploaded, split into the training and evaluation subsets and then used to fit the neural network. The data set used for training the model is dataset DGTA-BENCH which is available through TensorFlow datasets API and used for training other neural network architectures. It is a collection of Domain Generation Algorithms (DGA), Domain Tunneling Algorithms (DTA), and legitimate DNS names used to access popular Internet resources, safe domains hosted by DNS providers, as well as domains of Content Delivery networks (CDN). It contains 1.65M labelled domain names divided into 55 classes, 4 of which are DTA, 50 of which are DGA and one legitimate class.

2. *DNS Amplification attack*

a. We have tried to limit the number of queries per client at the relay’s end. Our protocol works on top of TCP, preventing source IP spoofing.

b. The relay, although unknown of the actual query/answer inside the encrypted packets, keeps a count of the number of queries sent by a particular client (IP address) and prevents from forwarding a query to the resolver if the count gets past a certain threshold.

3. *DNS Replay attack*

    a. If an attacker captures the query, he won't be able to resend it as it will be encrypted by the symmetric key of that client.

    b. Moreover, the attacker will have to encrypt it again with its own symmetric key to send it to the resolver. Our setting prevents *replay attacks* by default.

Active Issues

1. We are getting an incorrect response for some of the queries. We are working on improving the recursive resolver for accurately doing the resolution process.

2. The ML model we are using gives false positives. The accuracy of the model is something which we are eagerly looking at. More training data will help our model give better results.

Challenges Faced

1. Initially, we started with the client and the resolver and later implemented the relay. The client was not receiving any response from the resolver. We looked at the DNS packet to see if it was formed correctly or not. Still, we were not getting any close to the solution. Then, we learnt about this tool, *Wireshark,* which is a widely-used network protocol analyser. We set up two virtual machines and ran resolver and client in each one. Then we captured the traffic using *Wireshark* and learned that the packet was malformed. Then, we tried to debug the code, which was a very simple mistake of pointers. We had put the question in the wrong place in the DNS packet, which created all this issue.

2. There was very little information available on how to prevent different DNS attacks; for instance, the amplification attack required us to use machine learning which we are unfamiliar with. Acquiring the dataset and using the correct and efficient model was quite challenging. This was in python, and the other part of our project is in C++, made integrating them challenging.