

CSE3002- Internet and Web Programming

Analysis and Audit

Embedded Project



DNS AUTHENTICATION SERVER

Fall Semester – 2022-2023

L43+44Slot

Submitted by:

Naman Mohan Paul - 20BCE2423

Anshdeep Singh Kapula - 20BCE0797

Akshat Bisht - 20BCE0629

Akshay Rai - 20BCE0617

Submitted to:

Mrs. Swetha N G

Abstract

1.1. Background:

DNS was designed in the 1980s when the Internet was much smaller, and security was not a primary consideration in its design. As a result, when a recursive resolver sends a query to an authoritative name server, the resolver has no way to verify the authenticity of the response. The resolver can only check that a response appears to come from the same IP address where the resolver sent the original query. But relying on the source IP address of a response is not a strong authentication mechanism, since the source IP address of a DNS response packet can be easily forged, or spoofed. As DNS was originally designed, a resolver cannot easily detect a forged response to one of its queries. An attacker can easily masquerade as the authoritative server that a resolver originally queried by spoofing a response that appears to come from that authoritative server. In other words, an attacker can redirect a user to a potentially malicious site without the user realizing it.

1.2. Objectives of the Project:

The main objective of the project is to provide a DNS attack free environment to the user such that he/she can surf on the internet without having to face attacks like DNS cache poisoning or DNS spoofing by providing the user with the authentication server which is simple to implement and thereby would provide the IP address of verified websites and block unwanted or malicious IP address requests.

1.3 Application of the Project:

1.3.1. The DNS Client is capable of resolving the IP address of a host from the host's name. It does this by sending DNS requests to a DNS Server. The IP address of a DNS Server is specified in the network

interface configuration file or can be obtained from the DHCP Server for the Local Area Network.

1.3.2. The DNS Client is capable of resolving the IP address of a host from the host's name. It does this by sending DNS requests to a DNS Server. The IP address of a DNS Server is specified in the network interface configuration file or can be obtained from the DHCP Server for the Local Area Network.

1.3.3. The DNS Client caches the resolved IP addresses. The length of time the resolved host IP address is kept in the local cache depends on the Time to Live (TTL) timeout. This is returned in an answering packet from the DNS Server. The next time a DNS is requested, the cache table is checked first. If a, the valid host is found IP address is resolved from the cache and no actual DNS request is sent to the DNS Server.

1.4. Overview:

This project aims at creating an authentication system that will detect any DNS spoofing or DNS cache poisoning and prevent the server from returning an incorrect result record or IP Address. This would thereby provide a secure server for the host to surf on the internet without any hassle related to any kind of DNS attacks, hence making the system robust.

Introduction

Everyone needs security in life whether or not it's physical or digital. These days, spoofing of IP addresses or DNS spoofing area units is used widely to hack one's system and steal info or modification the data. It's additionally victimization for cyber warfare to unfold the viruses within the systems,

networks, etc. the country's most status documents may be steel through it. For

ex. In 2006, unknown hackers dole out a serious DNS spoofing attack – the primary of its kind – against 3 native banks in Everglade State. The attackers went into the System and hacked the server's supplier of internet of 3 websites and rerouted traffic to its login page to extract important information. This has allowed them to gather an associate unrevealed variety of MasterCard numbers

and PINs alongside different personal info happiness to their house owners.

This happened again in 2015 as hackers used DNS spoofing and send the users that were using Asian nation airlines to another webpage which was showing "404" with a picture of the plane. though no knowledge was purloined or compromised throughout the attack, it blocked access to the website and flight standing checks for many hours. So, to prevent DNS spoofing, we tend to scan use the DNS table for bar of it. Our plan to use the DNS shopper table is additionally superb and necessary for bar of spoofing. It maintains the record of

real websites and assigns a key and string challenge thereto as a result of in future if anyone accesses it then if it's real then it attests the permission to the user to use it otherwise it record it as pretend. In it, we've got to create the algorithmic rule once at that time it runs mechanically for each kind of DNS.

Literature Review

SNo	Reference	Problems	Solution Proposed
1.	<i>DNS Security: Antonio Lioy, Fabio Maino, Marius Marian, Daniele Mazzocchi, Dipartimento di Automatica e Informatica Politecnico di Torino Torino (Italy) (Research Paper)</i>	DNS security	The present research paper is an overview of the security problems affecting the Domain Name System and also of the solutions developed throughout the last years in order to provide a better, trustworthy, and safer name resolution protocol for the expanding Internet community of present days. The paper discusses the basic notions regarding DNS and introduces the reader to the known security threats regarding DNS. The DNSSEC subset proposed is presented and analyzed from both theoretical and practical points of view, explaining the existing security features of the implementations available today.
2.	<i>Security System for DNS using Cryptography: Sachin Kumar Sinha, Avinash Kant Singh, Amaresh Sharma B.TECH(IT), ITM Gida Gorakhpur, GBTU (Research Paper)</i>	Secure system for DNS client	The mapping or binding of IP addresses to hostnames became a major problem in the rapidly growing Internet and the higherlevel binding effort went through different stages of development up to the currently used Domain Name System (DNS). DNS Security is designed to provide security by combining the concept of both the Digital Signature and Asymmetric key (Public key) Cryptography. Here the Public key is sent instead of the Private key. The DNS security uses Message-Digest Algorithm to compress the Message (text file) and PRNG(Pseudo Random Number Generator) Algorithm for generating Public and Private keys. The message combines with the Private key to form a Signature using DSA Algorithm, which is sent along

			with the Public key. The receiver uses the Public key and DSA Algorithm to form a Signature. If this Signature matches with the Signature of the message received, the message is Decrypted and read else discarded.
3.	<i>A Comprehensive Analysis of Spoofing by P. Ramesh Babu, D. Lalitha Bhaskari, and CH. Satyanarayana</i>	IP Address, Email, web spoofing	This paper will not discuss any solutions to the problems related to the Spoofing attacks but gave us an idea of how spoofing attacks are performed in real-time. The main plan of this paper was to teach about spoofing attacks. Spoofing suggests that impersonating another person or pc, typically by providing false info (E-mail name, computer address, or science address). It explains that Spoofing will withstand several forms within the applied science world, all of that involves some kind of false illustration of data. There are a unit a spread of strategies and kinds of spoofing. The paper introduces and explains the subsequent spoofing attacks science, E-Mail, Web, and DNS spoofing. There are not any legal or constructive uses for implementing spoofing of any kind. A number of the outcomes can be sport, theft, vindication, or another malicious goal. The magnitude of those attacks are often terribly severe; will value U.S.A. legion greenbacks. This Paper describes concerning varied spoofing sorts and provides a tiny low read on detection and hindrance of spoofing attacks.
4.	<i>DNS Spoofing in Local Networks Made Easy Nikhil Tripathi, Mayank Swarnkar, and Neminath Hubballi</i>	Targeted DNS spoofing attack	DNS poisoning has become difficult to launch due to the introduction of techniques like source port and query identification value randomization. In this paper, the authors proposed in

			<p>which a targeted DNS spoofing attack that exploits a vulnerability present in DHCP server-side IP address conflict detection technique. In this paper, it is shown that the proposed attack is easier to launch and requires minimal bandwidth as compared to previously known attacks. This paper also discusses how proposed attack can target even a single victim client also without affecting other clients. We test the effectiveness of proposed attack in a real network setup and report the results. Further, it is discussed how known detection and mitigation techniques are unable to detect the attack</p>
--	--	--	---

Drawbacks in Existing system

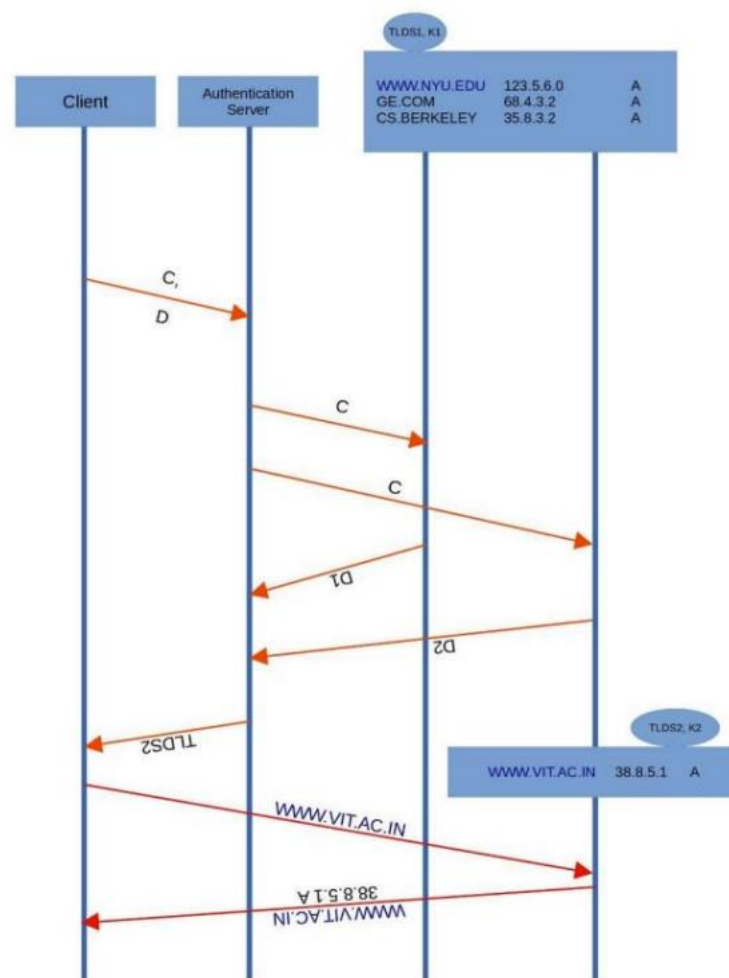
There are a unit principally 2 strategies by that DNS spoofing is disbursed – DNS cache poisoning and DNS ID spoofing. In DNS cache poisoning, the native DNS server is replaced with a compromised DNS server containing custom-made entries of real website names with the attacker's own information science addresses. Hence, once asking is shipped to the native DNS server for information science resolution, it communicates with the compromised DNS server, leading to the user being redirected to a false website planted by the wrongdoer. In DNS ID spoofing, the packet ID and information science data generated for the resolve request sent by the consumer is duplicated with false data within it. because the response ID matches the request ID, the consumer accepts the response containing the knowledge that's not expected. Common tips to stop DNS Spoofing embrace maintaining the DNS code up-to-date, maintaining separate servers for public and internal services, and

victimization secure keys to sign updates received from different DNS servers to avoid updates from non-trusted sources. Modern net browsers area unit designed by default to cache DNS records for a collection quantity of your time. the aim here is obvious; the nearer the DNS caching happens to the online browser, the fewer process steps should be taken to ascertain the cache and create the right requests to an information science address. once asking is created for a DNS record, the browser cache is that the initial location checked for the requested record. In Chrome, you'll be able to see the standing of your DNS cache by about to `chrome://net-internals/#dns`.

Proposed System

The DNS client will read from a file (INPUT-HNS.txt). Each line will contain a key, a challenge, and a hostname (6X\$7Q secretkey www.vit.ac.in). Combine key with the challenge to form digest for DNS client. Then connects to the AS server and sends ONLY the challenge string and the digest to the AS server. As server gives the string to TLDS servers and takes digests that are being formed by them. Then it does a digest compare and sends the name of the TLDS server that returns the correct digest. Note: as the keys read by the TLDS servers are different, only one digest will match. The DNS client then sends the hostname part of the query string to that TLDS server returned by the AS server and writes the response received from the TLDS server and the hostname of the TLDS server to the output file RESOLVED.txt. DNS table is being maintained by TLDS servers. The table contains 3 components that are hostname, IP address, flag. They also maintain a key for forming digest when they receive a challenge from AS server. DNS connects with TLDS server and sends hostname in form of string. The TLDS server does a lookup in the DNS_table and if there is a match, sends the DNS table entry as a string ["Hostname IP address A"]. If the hostname does not exist, then an error string Error: HOST NOT FOUND is returned. Note, that in this Project, the DNS client connects to the TLDS server to get the IP address for a given hostname. The hostname string, the Key, and

the challenge will be given one per line in a file in a text file (INPUTHNS.txt) and the keys one per line will be in a file INPUT-KEY1.txt and INPUT-KEY2.txt. The DNS table entries will also be one per line and will be in PROJ- TLDS1.txt and PROJ-TLDS2.txt. The TLDS servers, in addition to reading the DNS entries, will each obtain its key by reading the value from the corresponding key files. TLDS1 will read the key from INPUT-KEY1.txt and TLDS2 will read the key from INPUT-KEY2.txt The client program will output the results to a file RESOLVED.txt.



The client sends the query C and the digest to the TLDS server. The authentication server acts as a layer in between to ensure the authenticity of the response. The query C is sent to both the top level domain servers. They

return different responses with respect to their keys and the authentication server checks the correct response. The correct response is returned back to the client and the hostname is converted into the correct IP address. In the above case, the response from TLDS1 is rejected and TLDS2 is accepted.

Manual Demonstration for single input:

Here is a sample input line:

“K123 string www.facebook.com”

Challenge string and digest sent by the client to the Authentication server are string and c8c5230843ec953882c5724701e62cc8 respectively.

The authentication server then sends the Challenge string to both the top-level-domain servers: string

Response string sent from the TLDS server back to the AS server
c8c5230843ec953882c5724701e62cc8

Then DNS client sends the hostname part of the query string to that TLDS server and TLDS server will return the output:

www.facebook.com 192.64.4.2 A

Implementation

6.1. Discussion

We aim create a program so as to provide a DNS attack free environment to the user such that he/she can surf on the internet without having to face attacks like DNS cache poisoning or DNS spoofing by providing the user with the authentication server which is simple to implement and thereby would provide the IP address of verified websites and block unwanted or malicious IP address requests. The program would initiate the server from the client side send back the website addresses along with their authentic IP addresses and returning NOT FOUND message for those websites which either do not exist or those which are means of any DNS attack to the user and we can differentiate based on A for authenticated and NA for not authenticated.

6.2 Software Requirement

The Authentication Server would be using the socket header file in Python to create the server which would request for packets (IP addresses) from websites thereby detecting whether or not any DNS attack is taking place or not

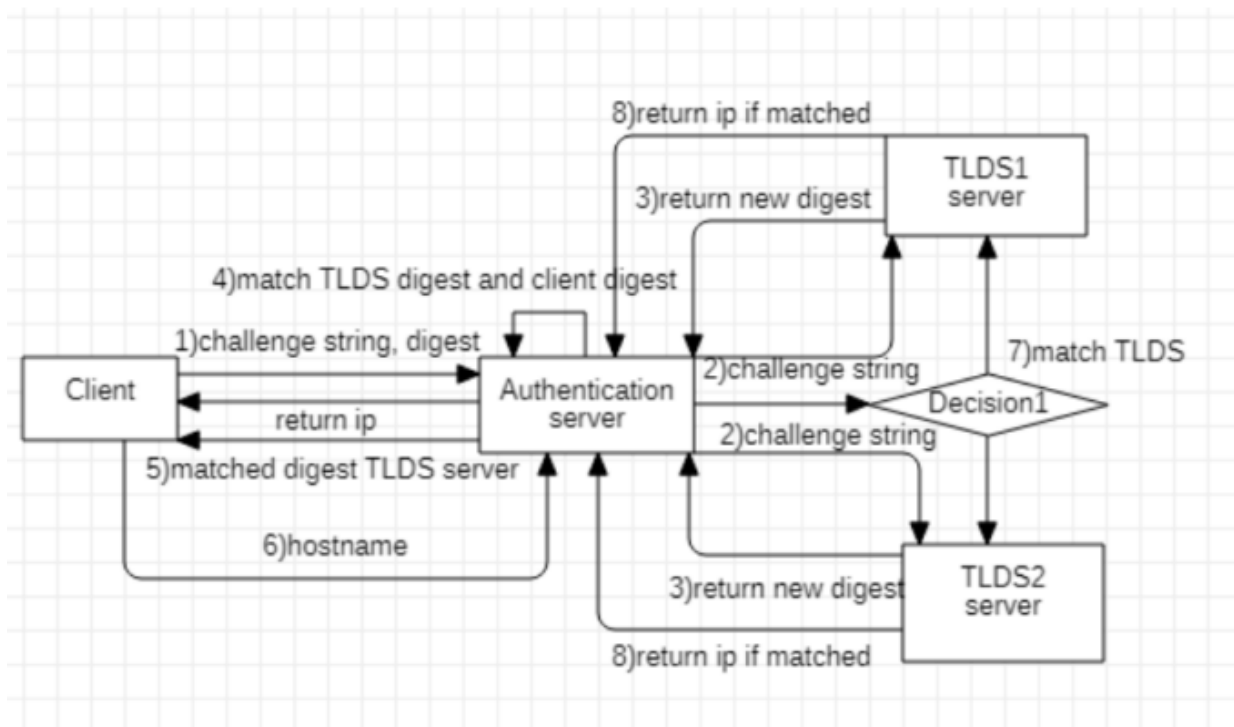
Software used: -

VS code

Programming Language used: -

Python

6.3 Architecture



6.4. Source Codes:

as.py (Authentication Server):

```

#RS server
import socket as mysoc
import pickle
import sys
import hmac
import hashlib

tlds1 = "TLDS1" #sys.argv[1]
tlds2 = "TLDS2" #sys.argv[2]

def rs():
    #initialize AS socket
    try:
        asssd = mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

    except mysoc.error as err:
        print('[RS]{} \n'.format("RS server socket open error ", err))

    #[tlds1 socket]
    try:
        astotscom=mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)
  
```

```

except mysoc.error as err:
    print('{} \n'.format("socket open error ", err))

#[tlds2 socket]
try:
    astotsedu=mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

except mysoc.error as err:
    print('{} \n'.format("socket open error ", err))

AS_table = {}
host=mysoc.gethostname()
com = tlds1
AS_table[com] = {'ip': mysoc.gethostbyname(host), 'flag':'NS'}
edu = tlds2
AS_table[edu] = {'ip': mysoc.gethostbyname(host), 'flag':'NS'}

try:
    sa_sameas_myaddr = AS_table[com]['ip']
    portTLDS1 = 5678
    server_binding = (sa_sameas_myaddr, portTLDS1)
    astotscom.connect(server_binding)
except mysoc.error as err:
    print('{} \n'.format("TLDS1 connect error "), err)
    exit()

try:
    sa_sameas_myaddr = AS_table[edu]['ip']
    portTLDS2 = 5677
    server_binding = (sa_sameas_myaddr, portTLDS2)
    astotsedu.connect(server_binding)
except mysoc.error as err:
    print('{} \n'.format("TLDS2 connect error "), err)
    exit()

server_binding = ('',50008)
assd.bind(server_binding)
assd.listen(1)
host=mysoc.gethostname()
print("[S]: Authentication Server host name is: ",host)
localhost_ip=(mysoc.gethostbyname(host))
print("[S]: Attempting to connect to client.\n[S]: Server IP address
is ",localhost_ip)
casd,addr=assd.accept()
print ("[S]: Got a connection request from a client at", addr)

server=""
#Authentication Loop

```

```

while True:
    data=casd.recv(100)
    if not data: break
    digest=pickle.loads(data)
    key = digest[0]
    challenge = digest[1]
    dvalid=hmac.new(key.encode(),challenge.encode("utf-8"),hashlib.sha1)
    astotscom.send(pickle.dumps(challenge))
    digestTLDS1=pickle.loads(astotscom.recv(100))
    astotsedu.send(pickle.dumps(challenge))
    digestTLDS2=pickle.loads(astotsedu.recv(100))
    if dvalid.hexdigest()==digestTLDS1:
        server="TLDS1"
        astotscom.send(pickle.dumps("pass"))
        astotsedu.send(pickle.dumps("fail"))
    elif dvalid.hexdigest()==digestTLDS2:
        server="TLDS2"
        astotscom.send(pickle.dumps("fail"))
        astotsedu.send(pickle.dumps("pass"))
    else:
        server="error"
        astotscom.send(pickle.dumps("fail"))
        astotsedu.send(pickle.dumps("fail"))
    casd.send(pickle.dumps(server))

# close everything
astotscom.close()
astotsedu.close()
assd.close()

rs()

```

client.py:

```

#Client
import socket as mysoc
import pickle
import sys
import time

def client():
    try:
        ctoas=mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

    except mysoc.error as err:
        print('{ } \n'.format("socket open error ",err))

```

```

#[tlds1 socket]
try:
    ctots1=mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

except mysoc.error as err:
    print('{} \n'.format("socket open error ", err))

#[tlds2 socket]
try:
    ctots2=mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

except mysoc.error as err:
    print('{} \n'.format("socket open error ", err))
try:
    file_name = "MIHIR-HNS.txt"
    fr = open(file_name, "r")
except IOError as err:
    print('{} \n'.format("File Open Error ",err))
    print("Please ensure table file exists in source folder")
    exit()
host=mysoc.gethostname()
sa_sameas_myaddr = mysoc.gethostbyname(host)
portAS = 50008
portTLDS1 = 5679
portTLDS2 = 5676
#[bind ctors socket to RS address, RS port]
try:
    server_binding=(sa_sameas_myaddr,portAS)
    ctoas.connect(server_binding)
except mysoc.error as err:
    print('{} \n'.format("connect error "), err)
    exit()
first1 = True
if first1:
    first1=False
    try:
        server_binding=(sa_sameas_myaddr,portTLDS1)
        ctots1.connect(server_binding)
    except mysoc.error as err:
        print('{} \n'.format("connect error "), err)
        exit()
first2 = True
if first2:
    first2=False
    try:
        server_binding=(sa_sameas_myaddr,portTLDS2)
        ctots2.connect(server_binding)
    except mysoc.error as err:

```



```

        print('{} \n'.format("connect error "), err)
        exit()
    with open("RESOLVED.txt", "w") as fw:
        for hostname in fr:
            digest = hostname.split()
            ctoas.send(pickle.dumps(digest))
            dataFromAS=pickle.loads(ctoas.recv(100))
            if "TLDS1" in dataFromAS:
                ctots1.send(pickle.dumps(digest[2]))
                data=pickle.loads(ctots1.recv(100))
                fw.write(data+'\n')
            elif "TLDS2" in dataFromAS:
                ctots2.send(pickle.dumps(digest[2]))
                data=pickle.loads(ctots2.recv(100))
                fw.write(data+'\n')
            else:
                fw.write(digest[2]+" - No servers matching key\n")
    # close everything
    time.sleep(5)
    fr.close()
    fw.close()
    ctots1.close()
    ctots2.close()
    ctoas.close()
    exit()

client()

```

tlds1.py:

```

#TS server
import socket as mysoc
import pickle
import sys
import hmac
import hashlib

#file_name = sys.argv[1]

def ts():
    try:
        astssd = mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

    except mysoc.error as err:
        print('[TS]: {} \n'.format("socket open error ", err))

```

```

try:
    ctssd = mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

except mysoc.error as err:
    print('[TS]: {}'.format("socket open error ", err))

try:
    file_TABLE = "MIHIR-TLDS1.txt"
    frTABLE = open(file_TABLE, "r")
except IOError as err:
    print('{} \n'.format("File Open Error ",err))
    print("Please ensure table file exists in source folder")
    exit()

TS_table = {}
for line in frTABLE:
    entry = line.split(' ')
    formatted_entry = []
    for item in entry:
        if item != ' ' and item != '':
            if item.endswith('\n'):
                item = item[:-1]
            formatted_entry.append(item)

    if formatted_entry[0] not in TS_table:
        TS_table[formatted_entry[0]] = {}
        TS_table[formatted_entry[0]]['ip'] = formatted_entry[1]
        TS_table[formatted_entry[0]]['flag'] = formatted_entry[2]

try:
    file_KEY = "MIHIR-KEY1.txt"
    frKEY = open(file_KEY, "r")
except IOError as err:
    print('{} \n'.format("File Open Error ",err))
    print("Please ensure key file exists in source folder")
    exit()

key = ""
for line in frKEY:
    key = line.strip()
    break

if not key:
    print("No key found")
    exit()

server_binding=('',5678)
astssd.bind(server_binding)
astssd.listen(1)
host=mysoc.gethostname()

```

```

print("[S]: Server host name is: ",host)
localhost_ip=(mysoc.gethostbyname(host))
print("[S]: Attempting to connect to as.\n[S]: Server IP address
is ",localhost_ip)
assd,addr=astssd.accept()
print ("[S]: Got a connection request from a client at", addr)

server_binding=('',5679)
ctssd.bind(server_binding)
ctssd.listen(1)
host=mysoc.gethostname()
print("[S]: Server host name is: ",host)
localhost_ip=(mysoc.gethostbyname(host))
print("[S]: Attempting to connect to as.\n[S]: Server IP address
is ",localhost_ip)
ctsd,addr=ctssd.accept()
print ("[S]: Got a connection request from a client at", addr)

while True:
    data = (assd.recv(100))
    if not data: break
    challenge=pickle.loads(data)
    digest = hmac.new(key.encode(),challenge.encode("utf-8"),hashlib.sha1)
    assd.send(pickle.dumps(digest.hexdigest()))
    auth=pickle.loads(assd.recv(100))
    if "fail" in auth:
        continue

    hnstring=pickle.loads(ctsd.recv(100))
    if not hnstring: continue
    entry = ''
    if hnstring in TS_table:
        entry = hnstring + ' ' + TS_table[hnstring]['ip'] + ' ' +
TS_table[hnstring]['flag']
    else:
        entry = hnstring + " - Error:HOST NOT FOUND"
    ctsd.send(pickle.dumps(entry))
frTABLE.close()
frKEY.close()
astssd.close()
ctssd.close()

ts()

```

tlds2.py:

```

#TS server
import socket as mysoc
import pickle
import sys
import hmac
import hashlib

#file_name = sys.argv[1]

def ts():
    try:
        astssd = mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

    except mysoc.error as err:
        print('[TS]: {}'.format("socket open error ", err))

    try:
        ctssd = mysoc.socket(mysoc.AF_INET, mysoc.SOCK_STREAM)

    except mysoc.error as err:
        print('[TS]: {}'.format("socket open error ", err))

    try:
        file_TABLE = "MIHIR-TLDS2.txt"
        frTABLE = open(file_TABLE, "r")
    except IOError as err:
        print('{} \n'.format("File Open Error ",err))
        print("Please ensure table file exists in source folder")
        exit()

    TS_table = {}
    for line in frTABLE:
        entry = line.split(' ')
        formatted_entry = []
        for item in entry:
            if item != ' ' and item != '':
                if item.endswith('\n'):
                    item = item[:-1]
                formatted_entry.append(item)

        if formatted_entry[0] not in TS_table:
            TS_table[formatted_entry[0]] = {}
            TS_table[formatted_entry[0]]['ip'] = formatted_entry[1]
            TS_table[formatted_entry[0]]['flag'] = formatted_entry[2]

    try:

```

```

        file_KEY = "MIHIR-KEY2.txt"
        frKEY = open(file_KEY, "r")
    except IOError as err:
        print('{ } \n'.format("File Open Error ",err))
        print("Please ensure key file exists in source folder")
        exit()
    key = ""
    for line in frKEY:
        key = line.strip()
        break

    if not key:
        print("No key found")
        exit()
    server_binding=('',5677)
    astssd.bind(server_binding)
    astssd.listen(1)
    host=mysoc.gethostname()
    print("[S]: Server host name is: ",host)
    localhost_ip=(mysoc.gethostbyname(host))
    print("[S]: Attempting to connect to as.\n[S]: Server IP address
is ",localhost_ip)
    assd,addr=astssd.accept()
    print ("[S]: Got a connection request from a client at", addr)

    server_binding=('',5676)
    ctssd.bind(server_binding)
    ctssd.listen(1)
    host=mysoc.gethostname()
    print("[S]: Server host name is: ",host)
    localhost_ip=(mysoc.gethostbyname(host))
    print("[S]: Attempting to connect to as.\n[S]: Server IP address
is ",localhost_ip)
    ctsd,addr=ctssd.accept()
    print ("[S]: Got a connection request from a client at", addr)

    while True:
        data = (assd.recv(100))
        if not data: break
        challenge=pickle.loads(data)
        digest = hmac.new(key.encode(),challenge.encode("utf-8"),hashlib.sha1)
        assd.send(pickle.dumps(digest.hexdigest()))
        auth=pickle.loads(assd.recv(100))
        if "fail" in auth:
            continue

        hnstring=pickle.loads(ctsd.recv(100))
        if not hnstring: continue

```

```

    entry = ''
    if hnstring in TS_table:
        entry = hnstring + ' ' + TS_table[hnstring]['ip'] + ' ' +
TS_table[hnstring]['flag']
    else:
        entry = hnstring + " - Error:HOST NOT FOUND"
    ctssd.send(pickle.dumps(entry))

frTABLE.close()
frKEY.close()
astssd.close()
ctssd.close()

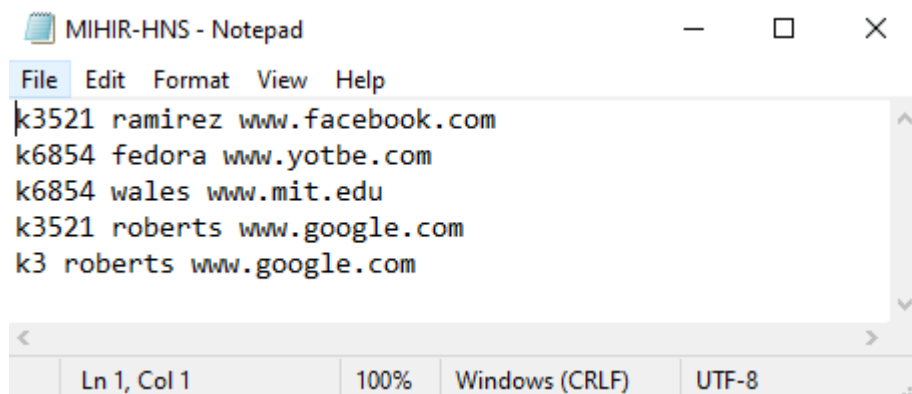
ts()

```

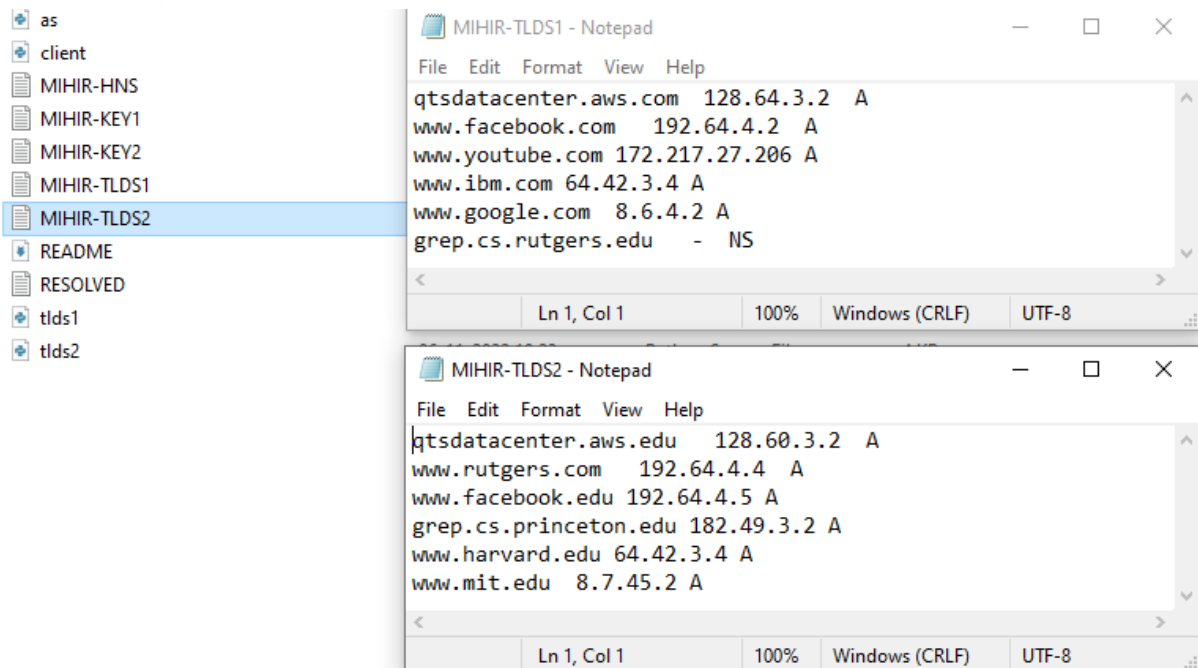
Result

Input:

First, start the two TLDS servers, then the AS server, and then the client program. This will start communication of the port number and hostname of TS servers to the AS server and hostname of the AS server to the client program. Data structure for the DNS table as well as to store KEYS and digests is a text file.



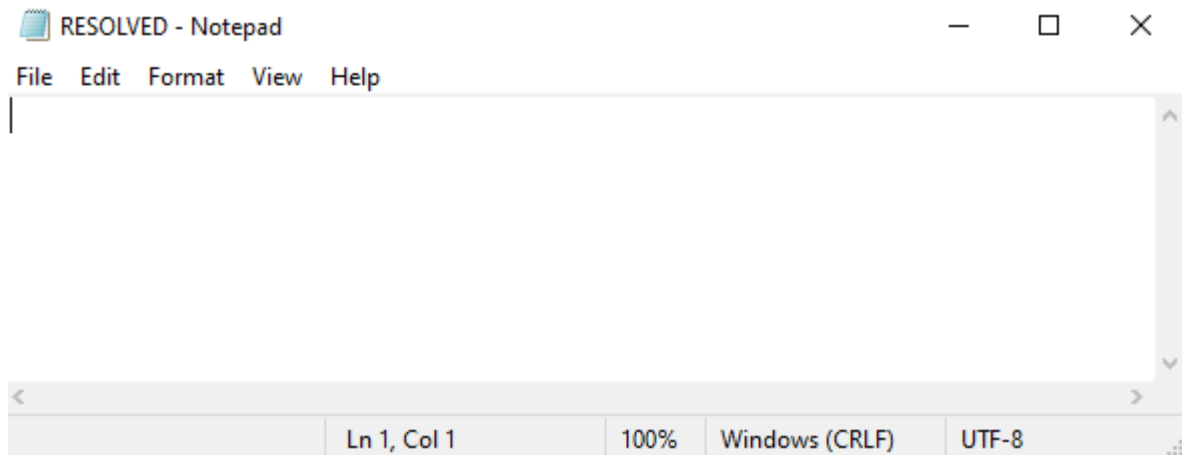
Input of the top level domain servers given:-



Two key files given :-

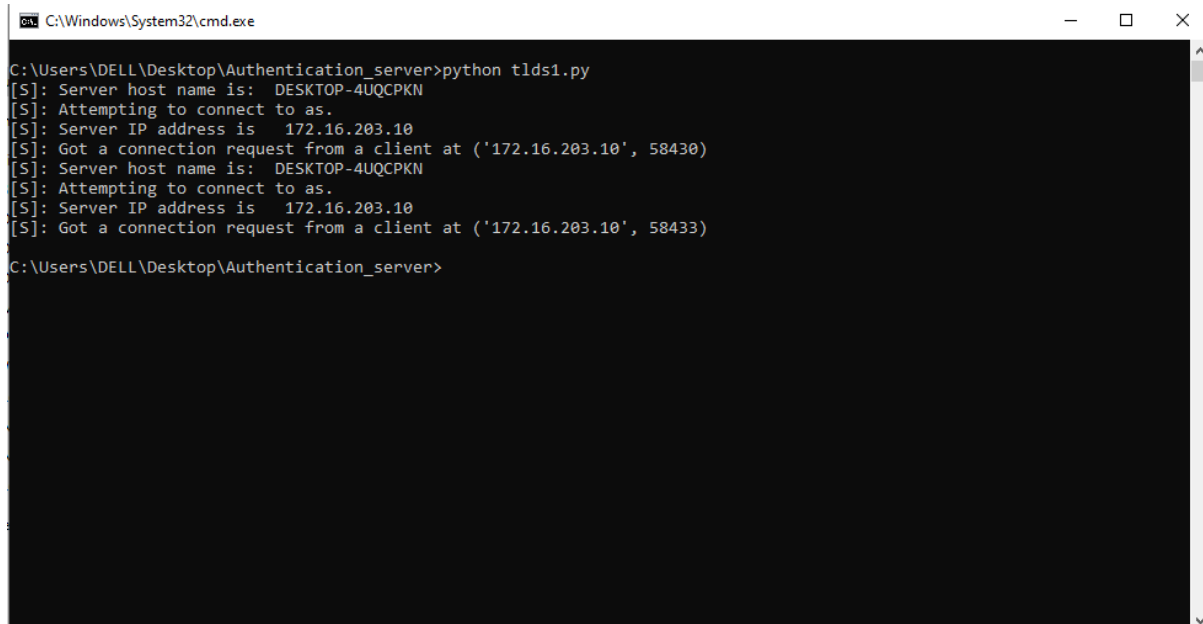
MIHIR-KEY1	17-10-2022 12:23	Text Document	1 KB
MIHIR-KEY2	17-10-2022 12:23	Text Document	1 KB

Empty resolved file as shown :-



Output:

Output for tlds1.py

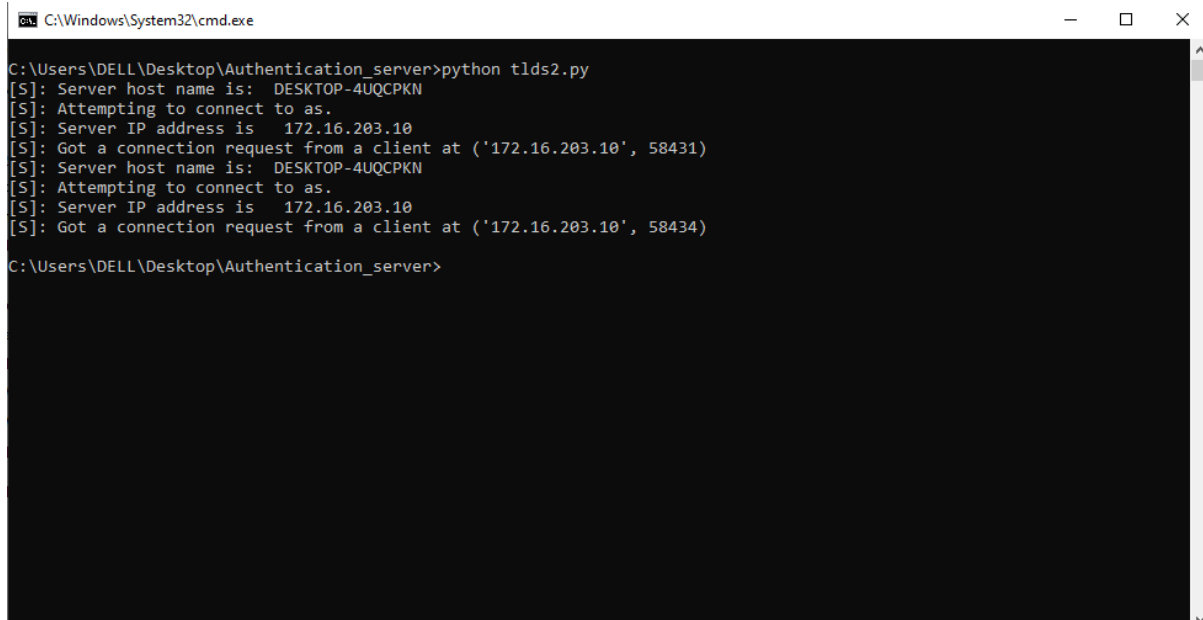


```
C:\Windows\System32\cmd.exe

C:\Users\DELL\Desktop\Authentication_server>python tlds1.py
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.16.203.10
[S]: Got a connection request from a client at ('172.16.203.10', 58430)
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.16.203.10
[S]: Got a connection request from a client at ('172.16.203.10', 58433)

C:\Users\DELL\Desktop\Authentication_server>
```

Output for tlds2.py

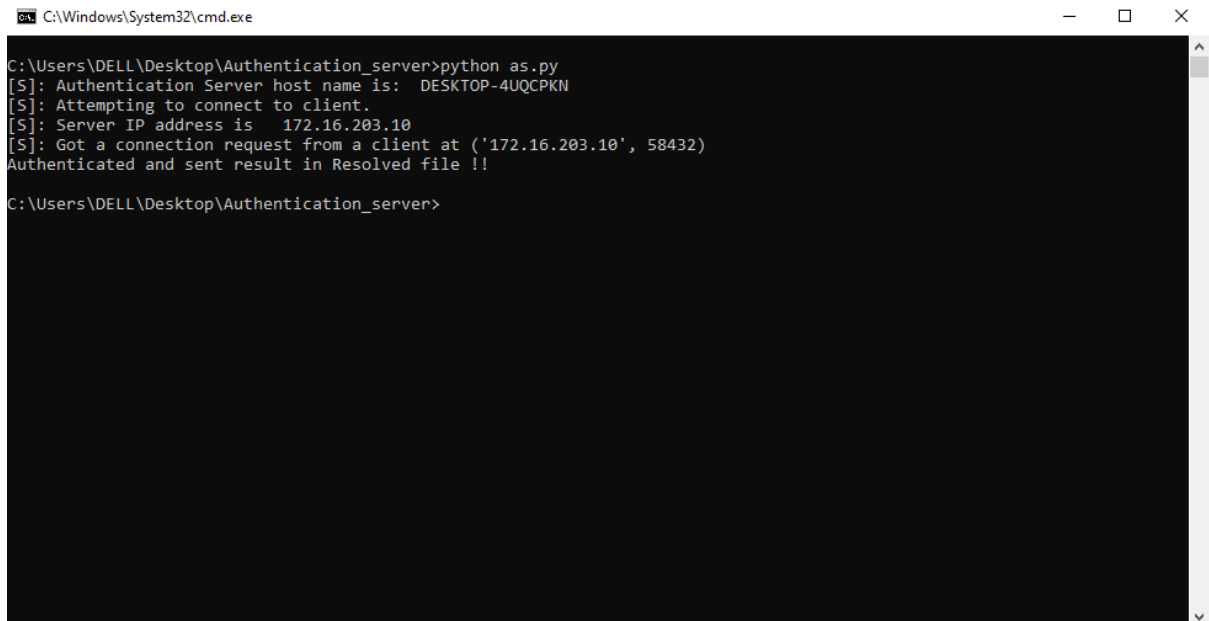


```
C:\Windows\System32\cmd.exe

C:\Users\DELL\Desktop\Authentication_server>python tlds2.py
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.16.203.10
[S]: Got a connection request from a client at ('172.16.203.10', 58431)
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.16.203.10
[S]: Got a connection request from a client at ('172.16.203.10', 58434)

C:\Users\DELL\Desktop\Authentication_server>
```

Output for as.py

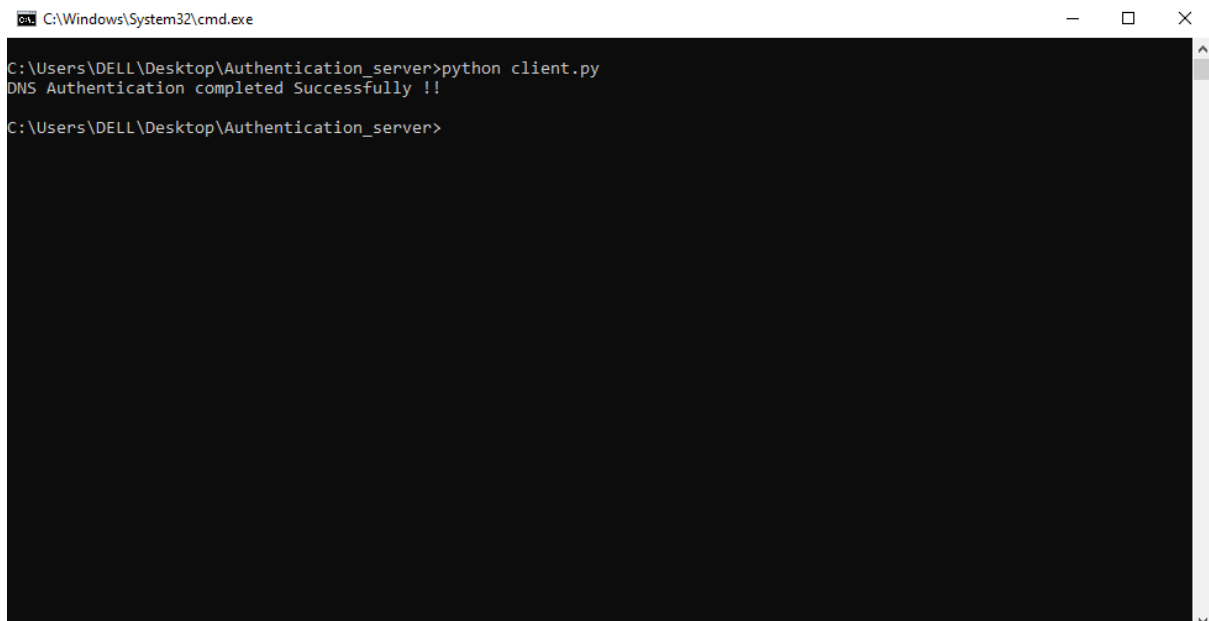


```
C:\Windows\System32\cmd.exe

C:\Users\DELL\Desktop\Authentication_server>python as.py
[S]: Authentication Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to client.
[S]: Server IP address is 172.16.203.10
[S]: Got a connection request from a client at ('172.16.203.10', 58432)
Authenticated and sent result in Resolved file !!

C:\Users\DELL\Desktop\Authentication_server>
```

Output for client.py



```
C:\Windows\System32\cmd.exe

C:\Users\DELL\Desktop\Authentication_server>python client.py
DNS Authentication completed Successfully !!

C:\Users\DELL\Desktop\Authentication_server>
```

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL\Desktop\Authentication_server>python tlds1.py
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.20.10.2
[S]: Got a connection request from a client at ('172.20.10.2', 64000)
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.20.10.2
[S]: Got a connection request from a client at ('172.20.10.2', 64003)
C:\Users\DELL\Desktop\Authentication_server>

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL\Desktop\Authentication_server>python as.py
[S]: Authentication Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to client.
[S]: Server IP address is 172.20.10.2
[S]: Got a connection request from a client at ('172.20.10.2', 64002)
Authenticated and sent result in Resolved file !!
C:\Users\DELL\Desktop\Authentication_server>

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL\Desktop\Authentication_server>python tlds2.py
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.20.10.2
[S]: Got a connection request from a client at ('172.20.10.2', 64001)
[S]: Server host name is: DESKTOP-4UQCPKN
[S]: Attempting to connect to as.
[S]: Server IP address is 172.20.10.2
[S]: Got a connection request from a client at ('172.20.10.2', 64004)
C:\Users\DELL\Desktop\Authentication_server>

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL\Desktop\Authentication_server>python client.py
DNS Authentication completed Successfully !!
C:\Users\DELL\Desktop\Authentication_server>

```

Outcome

Resolved file is created on time of creation and can be seen over here. Even if it is present already in an empty state and the code is run then all the resolved ip address along with the domain name are given as an output in this file as shown:

```

RESOLVED - Notepad
File Edit Format View Help
www.facebook.com 192.64.4.2 A
www.yotbe.com - Error:HOST NOT FOUND
www.mit.edu 8.7.45.2 A
www.google.com 8.6.4.2 A
www.google.com - No servers matching key
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

As a result we were able to implement our model successfully with the help of python. Thus an authentication server was established that ensured that the user doesn't become a victim of DNS spoofing of DNS cache poisoning.

Future Work

We will try to automate this if it gets succeeded at ground level and take use of one of tlds servers from 1511 tlds servers around the world. Then we will be able to provide safe access to all users to legitimate sites so that they can surf on the network without fear of being directed to fake sites. Once recognised worldwide the net traffic control authorities can make every new coming and already existing websites to submit they domain name along with their IP address on this tlds server, every coming entry would be first checked for authentication and then only stored in the tlds and host name providing spoofed IP address can be blocked permanently. Also any site refusing to send its IP address and host name to tlds should not be allowed to entertain any traffic. Thus by protecting the traffic from being forwarded to spoofed sites we will prevent many attacks like zombie, virus etc.

Conclusion

The system was successfully able to prevent DNS spoofing after it was tested on a wrong IP address it maintains a key which is used to create a digest using HMAC algorithm provided by the hmac module from python. With the right size of the key this can be a very effective method to prevent DNS spoofing. The system was tested against a fake host name to which the result was an error message. The problem of DNS spoofing can result in huge loss of valuable data which can be secured by just adding a authentication server as implemented by our system. The only requirement for our system is python language installed on the system. We believe DNS would make a good distribution point of application keys and certificates for large scale systems. The main reason is that DNS is a unique provider of bindings between commonly used names. We presented a proposal for DNSSEC that, when properly implemented, offers the highest level of security while reducing network traffic. In addition, it reduces storage requirements and enables efficient mutual authentication. In particular, for highly critical parts of the DNS, like root servers or other servers near the root, our service can provide increased security. We rely on DNSSEC to provide authenticated delegation,

while keeping the functional overhead of key distribution outside the critical DNS infrastructure. This strategy allows us to use the name service infrastructure to guarantee authenticity