DNS Steganography - Progetto Cyber Security

Ciro Giuseppe De Vita
Università degli Studi di Napoli "Parthenope"
cirogiuseppe.devita001@studenti.uniparthenope.it

***Abstract*—Questo progetto d'esame mostra la possibilità di usare il protocollo DNS (Domain Name System) per trasportare messaggi segreti (criptati) tra un computer e un server DNS compromesso.**

***Keywords*—DNS, Steganography, Covert Channel**

I. Introduzione

La tecnica più comune per trasferire informazioni sicure è la **crittografia**, che utilizza algoritmi per mascherare ciò che si vuole comunicare. Sebbene ciò renda le informazioni illeggibili da parte di terzi, non nasconde il fatto che la comunicazione sia in corso. I **covert channel** vengono utilizzati per tentare di nascondere l'esistenza di un canale di comunicazione. L'applicazione originale dei covert channel era quella di risolvere il problema dei *prigionieri*, in cui due parti vogliono comunicare ma la comunicazione è mediata da un guardiano che è in grado di leggere i messaggi e determinare se sono autorizzati. Dunque, devono escogitare un modo per nascondere la loro conversazione segreta in un modo che non sembri sospetto.
Con l'enorme quantità di traffico su Internet, i protocolli di rete sono diventati un veicolo comune per i covert channel, che in genere nascondono le informazioni negli **header fields** dei pacchetti.
La mia idea è quello di nascondere le informazioni segrete da inviare in queries DNS come mostrato in figura 1.
Uno dei protocolli più importanti per il funzionamento di

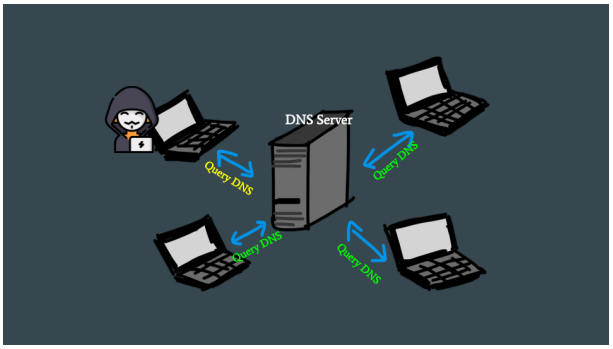

Fig. 1. Idea

Internet è il **DNS**. Il Domain Name System (DNS) è di vitale importanza per qualsiasi risorsa connessa a Internet o alla rete privata. Lo usiamo ogni volta che apriamo pagine Internet o controlliamo le nostre e-mail e sarà molto difficile immaginare l'architettura della moderna rete di comunicazione di oggi senza DNS. Quindi, è un buon candidato per nascondere informazioni al suo interno, considerando anche il fatto che le query DNS sono poco filtrate a causa della grande necessità di accesso a Internet.
Il DNS viene utilizzato per tradurre il nome mnemonico del server nel suo indirizzo IP corrispondente. Il DNS si basa sul protocollo **UDP**, il che significa che è **connectionless** e ha una bassa affidabilità.

A. Protocollo DNS

Il formato dei messaggi DNS è costante, indipendentemente dal tipo di richiesta. Un messaggio che contiene una risposta rispetto ad una domanda è più grande perché utilizza più campi. Tutti i pacchetti DNS hanno la seguente struttura:

1) **Header**: contiene informazioni di base che consentono di inviare e identificare i messaggi.
2) **Question**: contiene le query per un name server.
3) **Answer**: contiene le risposte delle queries.
4) **Authority**: indica i server autoritativi per un dominio.
5) **Additional**: dedicato per ulteriori informazioni.

B. Header

L'header contiene molti campi, ma i più importanti studiati per la soluzione presentata in questo progetto sono:

- **ID**: può essere visto come una chiave di autenticazione per ogni richiesta DNS e dovrebbe essere abbastanza casuale da assicurarsi che ogni query abbia un ID DNS univoco.
- **ANCount**: specifica il numero di answers.
- **QDCount**: specifica il numero di queries mandate.

C. Answer

Una query ed una answer DNS hanno strutture molto simili e i loro campi sono:

- **Name**: contiene il nome dell'oggetto, della zona o del dominio che identifica la query.
- **Type**: contiene il tipo di record. Il tipo più popolare è il record A, ovvero una query per ottenere l'indirizzo IPv4 del dominio specificato nel campo Nome. Rispettivamente, AAAA è una query per ottenere l'indirizzo IPv6.
- **Class**: definisce la classe di una query e di solito ha il valore "1" che è IN (Internet).
- **TTL**: serve a dire al server ricorsivo o al resolver locale per quanto tempo deve mantenere tale record nella sua cache. Più lungo è il TTL, più a lungo il resolver conserva le informazioni nella sua cache. Minore è il TTL, minore

è il tempo in cui il resolver conserva tali informazioni nella sua cache.
- **Resource Data**: contiene byte di dati. Ad esempio, per un record di tipo A (una query base per un DNS), sono richiesti quattro byte contenenti un indirizzo IPv4.

II. Stego-Sistema proposto

In questo capitolo effettuerò un'analisi steganografica sul meccanismo di come nascondere informazioni segrete all'interno di query DNS. In particolare, analizzerò due campi di una query DNS, ovvero il **DNS ID** e il campo **TTL** che è presente all'interno del campo *"Answer"*.
In particolare, il campo TTL sarà usato per indicare al mio server DNS infetto che si sta per inviare un messaggio contenente informazioni segrete, infatti servirà come ***start session*** della comunicazione. Invece, il campo DNS ID verrà usato per nescondere il vero e proprio messaggio segreto (fase di ***data exchange***).

A. Answer Field - TTL

Dalle specifiche del protocollo DNS si può notare che una query DNS può contenere anche risposte, infatti il protocollo non vieta l'inserimento di una risposta, ma lo consiglia solamente (***should***).

L'analisi teorica eseguita è stata confermata con test effettuati inviando query standard e preparate a un server DNS di Google (IP: 8.8.8.8), e seguendole in WireShark, un programma per il monitoraggio del traffico di rete. Dall'analisi sono state tratte le seguenti conclusioni:

1) Il server DNS che elabora le query ha ignorato le query distorte, ovvero le query contenenti header fields completati in modo non standard.
2) Anche le query DNS (messaggi con un flag QR impostato su 0) possono avere risposte; tale query non viene trattata come distorta.

Dunque, una volta avuta la conferma del possibile inserimento di una risposta all'interno di una query, per poter inserire e nascondere le informazioni di inizio comunicazione (start session) ho usato il campo TTL. Questo ha una grandezza di 16 bit in cui poter inserire le informazioni di inizio comunicazione, in particolare di questi 16 ne ho usati solamente 12 mentre i rimanenti sono stati generati casulamente.

B. DNS ID

Come spiegato in precedenza, il campo DNS ID di una query DNS può essere visto come una chiave di autenticazione e può essere usato per mandare un messaggio segreto. In particolare, si può sfruttare la sua natura *randomica* per inserire informazioni che poi il server DNS infetto intercetterà. Il DNS ID è composto da 16 bit e dunque, volendo, si potrebbero mandare 2 Byte (2 caratteri) per ogni query DNS, ma, come si può vedere nei capitoli successivi, non è stato fatto così.

III. Meccanismo Staganografico

L'algoritmo sviluppato per poter nascondere messaggi in query DNS può essere visto come composto da due fasi ben distinte:

- **Start Session**
- **Data Exchange**

Nella prima fase si vuole indicare al server DNS che si sta per trasmettere un messaggio segreto, mentre nella seconda fase c'è il vero invio del messaggio segreto nascoto in diverse query DNS.
I passi dell'algoritmo sono i seguenti:

1) Criptazione del messaggio da inviare
2) Calcolo lunghezza del messaggio
3) Incapsulazione del messaggio di **start session** all'interno del campo TTL
4) Divisione del messaggio in chunks da massimo 16 bytes ciascuno
5) Per ciascun chunk invio di 16 query DNS in cui sono incapsulati i singoli caratteri del messaggio (16 bytes equivalgono a 16 caratteri) nel campo DNS ID

A. Start Session

Vediamo nel dettaglio come ho incapsulato il messaggio nel campo TTL. Come detto in precedenza, la prima fase dell'algoritmo è quella di indicare al server DNS che si sta per iniziare una comunicazione.
Il campo TTL è composto da 16 bit e le informazioni sono state nascoste nel seguente modo:

- nei primi 4 bit pari ho inserito un **pattern**, che servirà al mio server DNS per controllare se prendere in considerazione la query;
- nei primi 8 bit dispari ho inserito la **lunghezza** del messaggio espressa in binario, che servirà per indicare quante query DNS aspettarsi;
- gli ultimi 4 bit pari, invece, sono stati generati casualmente.

Questa sequenza può essere osservata in Figura 2. Inoltre, si può notare che sono stati posti ulteriori due vincoli, ovvero il domain name di cui si vuole conoscere l'IP deve avere un numero di consonanti pari e almeno quattro vocali. Ciò è stato fatto per diminuire il numero di falsi positivi che il server DNS non dovrà considerare. Da notare che il domain name di cui si vuole conoscere l'indirizzo IPv4 viene selezionato casualmente da una lista contenente i 1000 domini più visitati in Internet, per rendere la comunicazione il più reale possibile.

B. Data Exchange

Terminata la prima fase, ora è possibile incapsulare il messaggio da inviare al server DNS infetto. Vediamo nel dettaglio come è stato possibile incapsulare il messaggio nel campo DNS ID.
Il messaggio viene prima criptato, poi diviso in chunks, ognuno composto da 16 bytes (ovvero 16 caratteri) ciascuno, e per ciascun chunk vengono effettuate 16 queries DNS. In

Fig. 2. Start Session

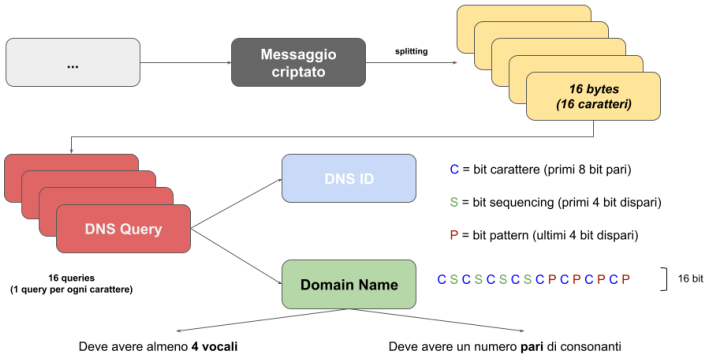

Fig. 3. Data Echange

ciascuna query è incapsulato un singolo carattere che compone il messaggio. Quest'incapsulazione avviene nel seguente modo:

- nei primi 8 bit pari è stato inserito il carattere ASCII corrispondente ed espresso in binario. Ad esempio la lettera 'a' corrisponde a 97 ed espressa in binario corrisponde a '01100001'.
- nei primi 4 bit dispari è stato inserito un valore di sequencing, ovvero un numero che identifica la posizione del carattere all'interno del chunk. Ciò è stato utile al server DNS per ricostruire il messaggio segreto inviato, poichè, essendo il DNS un protocollo UDP, non è detto che i pacchetti arrivino sequenzialmente.
- infine, negli ultimi 4 bit dispari è stato inserito il pattern, che serve per indicare al server DNS di prendere in considerazione la query. Da notare che il pattern è lo stesso di quello usato nello step precedente.

Questa sequenza può essere osservata in Figura 3. Inoltre, si può notare che, oltre al campo DNS ID, formattato nel modo appena descritto, anche in questo caso, come nel caso della fase di start session, il domain name di cui si vuole conoscere l'indirizzo IP deve avere almeno 4 vocali e un numero pari di consonanti. Anche in questo caso, ciò è stato fatto per diminuire il numero di falsi positivi.

C. Server Side

Nei paragrafi precedenti è stato descritto il meccanismo di incapsulazione del messaggio segreto e della fase di start session. In questo paragrafo, andremo a vedere come il server DNS intercetta le query e **"decripta"** il messaggio. Il server DNS è sempre in ascolto e ogni volta che riceve una query DNS effettua i seguenti controlli:

1) controlla se la query DNS contiene anche una risposta (*answer*): in caso affermativo, quella query potrebbe contenere il messaggio di start session. Dunque si va a leggere il campo TTL per ottenere il pattern che identifica la query e si verifica che il domain name abbia almeno 4 vocali e un numero pari di consonanti. Per ottenere il pattern, il campo TTL, che è un numero, viene convertito in binario e vengono estratti i primi 4 bit pari. Se sono verificate tutte le 3 condizioni, allora, significa che quella query deve essere presa in considerazione e si estra la lunghezza del messaggio, che identifica il numero di query DNS con un campo DNS ID modificato che il client invierà al server. Per ottenere la lunghezza, si estraggono sempre dal campo TTL i primi 8 bit dispari, ottenendo un numero binario che, trasformato in intero, rappresenta la lunghezza del messaggio.
Invece, nel caso che anche solo una condizione non dovesse verificarsi, la query DNS non viene presa in considerazione perchè sicuramente non contiene alcuna informazione utile.
2) se la query DNS non contiene anche una risposta, allora, questa query potrebbe contenere un carattere segreto nascosto nel campo DNS ID. Il primo controllo da effettuare è quello di controllare se in precedenza è stata ricevuta una query contenente il messaggio di start session, poichè, se così non fosse, allora tutte le query devono essere ignorate, perchè sicuramente non contengono alcuna informazione nascosta. In caso affermativo, inizia la fase di estrazione del carattere dal campo DNS ID. Questa fase è composta dai seguenti passi:
   a) per prima cosa si estraggono i primi 8 bit pari dal campo DNS ID, ottenendo un numero binario, quest'ultimo viene convertito in intero che a sua volta viene convertito in carattere ASCII. Se il carattere appartiene all'insieme dei caratteri della codifica in base64, ovvero se è compreso tra A-Z, a-z, 0-9, oppure è / o =, allora la query viene presa in considerazione e si procede con la "decodifica", altrimenti viene scartata perchè non può contenere alcuna informazione segreta.
   b) in seguito, si estraggono gli ultimi 4 bit dispari, sempre dal campo DNS ID, per ottenere il pattern di identificazione, poi, si controlla il campo domain name che deve avere almeno 4 vocali e un numero pari di consonanti. Se il pattern è lo stesso di quello

ottenuto nella fase di start session e sono verificate anche le condizioni sul domain name, allora si prosegue con la fase di decodifica, altrimenti la query non viene presa in considerazione.

c) infine, si va ad effettuare un ulteriore controllo per verificare se quella query è un falso positivo. Infatti, nonostante abbia messo un buon numero di vincoli, non è impossibile che ci possano essere delle query reali che rispettino i vincoli che ho imposto, generando, quindi, dei falsi positivi. Il controllo avviene nel seguente modo: dato che da lato client non invio mai due query consecutive contenenti caratteri segreti, controllo se la query precedente è stata considerata. In caso positivo, allora, scarto la query corrente e proseguo, altrimenti prendo in considerazione la query corrente e vado ad estrarre il carattere segreto (contenuto nei primi 8 bit pari).

3) la seconda fase avrà fine quando il numero di query DNS col campo DNS ID considerate avrà raggiunto la lunghezza del messaggio, ottenuta nella fase di start session.
4) infine, si costruisce il messaggio con tutti i caratteri contenuti nelle query DNS ”malevole” e lo si decripta.

Questa seguza può essere osservata in Figura 4.

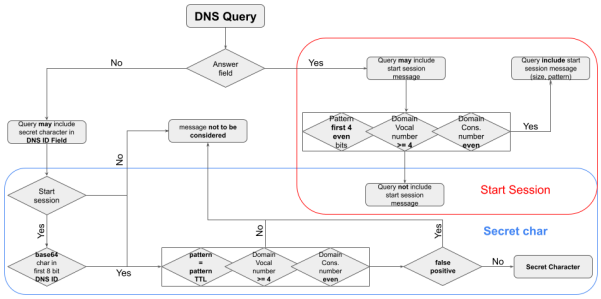

Fig. 4. Server Side

IV. Ambienti di sviluppo

Terminata la descrizione sul funzionamento dell'algoritmo, analizziamo nel dettaglio come è stato possibile realizzare ciò. Sia il client che il server sono stati scritti in Python, in particolare ho usato **Python 3.9**. Tale scelta è stata dovuta dal fatto che Python mette a disposizione delle librerie che permettono di intercettare e manipolare le queries DNS abbastanza facilmente.
Il server DNS è stato scritto da zero (*”from scratch”*) grazie all'utilizzo della libreria ***”dnslib”***. Questo server DNS non fa altro che ascoltare le queries DNS che riceve da diversi client e rigirarle (*”forward”*) ad un vero server DNS, in questo caso ho usato un Google DNS (8.8.8.8). Una volta ricevuta la risposta dal server DNS reale, la inoltra al client che aveva mandato la query. Dunque, il server DNS si comporta come un normale server DNS così da non destare sospetti. Ciò è mostrato nel seguente spezzone di codice.

```
from dnslib.proxy import ProxyResolver
from dnslib.server import DNSServer

if __name__ == '__main__':
    signal.signal(signal.SIGTERM, handle_sig)

    framestore = []
    port = int(os.getenv('PORT', 53))
    upstream = os.getenv('UPSTREAM', '8.8.8.8')
    resolver = Resolver(upstream)
    udp_server = DNSServer(resolver, port=port)
    tcp_server = DNSServer(resolver, port=port, tcp=
    True)

    logger.info('starting DNS server on port %d,
    upstream DNS server "%s"', port, upstream)
    udp_server.start_thread()
    tcp_server.start_thread()

    try:
        while udp_server.isAlive():
            sleep(1)
    except KeyboardInterrupt:
        pass
```

Listing 1. DNS Server

Invece, nel prossimo spezzone di codice, è mostrata l'implementazione dell'intercettazione delle queries e le politiche adottate per il riconoscimento e la decodifca dei messaggi segreti.

```
def resolve(self, request, handler):
    if request.a.rdata is not None and self.
    countConsonants(str(request.a.rname))[0] % 2 ==
    0 and self.countConsonants(str(request.a.rname))
    [1] >= 4:
        ttl = request.a.ttl
        binary = bin(ttl)[2:].zfill(16)
        final_binary = ''
        pattern = ''
        for i in range(0, len(binary)):
            if i % 2 != 0:
                final_binary += binary[i]
            elif i < 8 and i % 2 == 0:
                pattern += binary[i]

        if pattern == config.get('CONFIG', 'pattern'
    ):
            self.length = int(final_binary, 2)
            self.pattern = int(pattern, 2)
            self.framestore = [None] * self.length
            self.current_i = 0
            self.allFrames.append("no")
    else:
        if self.pattern != -1:
            dns_id = request.header.id
            binary = bin(dns_id)[2:].zfill(16)
            final_binary = ''
            sequence_number = ''
            pattern = ''
            for i in range(0, len(binary)):
                if i % 2 == 0:
                    final_binary += binary[i]
                elif i < 8 and i % 2 != 0:
                    sequence_number += binary[i]
                else:
                    pattern += binary[i]

            c = chr(int(final_binary, 2))
```

```
            search = re.compile(r'[A-Za-z0-9+/= ]').search

            if bool(search(c)):
                if int(pattern, 2) == self.pattern and self.countConsonants(str(request.q.qname))[0] % 2 ==  and self.countConsonants(str(request.q.qname))[1] >= 4:
                    if self.allFrames[-1] != "ok":
                        self.allFrames.append("ok")
                        self.number = int(sequence_number, 2)

                        try:
                            self.framestore[self.number + 16 * self.current_i] = c
                        except Exception as e:
                            print(e, self.number + 16 * self.current_i)
                            pass

                        if self.number == 15:
                            self.current_i += 1

                        if None not in self.framestore:
                            self.current_i = 0
                            self.pattern = -1
                            combined_payloads = ''.join(self.framestore)
                            self.save_on_file(combined_payloads)
                    else:
                        print("False Positive")
                        self.allFrames.append("no")
                else:
                    self.allFrames.append("no")
            else:
                self.allFrames.append("no")

        return super().resolve(request, handler)
```

Listing 2. Resolve function

Invece, l'invio e la manipolazione delle queries DNS lato client è stato implementato grazie all'ausilio della libreria ***scapy***. Lato client ho scelto di utilizzare questa libreria e non "dnslib" poichè con quest'ultima mi è risultato più difficile manipolare le query DNS non essendoci una documentazione ufficiale in cui spiegano come fare, a differenza della libreria scapy, dove c'è molta documentazione su come creare query DNS e inviarle. Dunque la libreria "dnslib" mi è stata utile per intercettare le queries e rigirarle ad un server DNS reale. Di seguito è mostrato un esempio della creazione di una query DNS contenente anche un campo "answer" tramite la libreria scapy.

```
sr1(IP(dst=server) / UDP(sport=RandShort(), dport=53) / DNS(id=random.randint(0, 65535), rd=1, qd=DNSQR(qname=fake_domain), an=DNSRR(rrname=fake_domain, rdata=ip)), verbose=0)
```

Listing 3. Example Scapy query DNS

Di seguito, invece, è mostrata la funzione lato client per mandare il messaggio segreto.

```
    def send_message(self, server, message):
        config = ConfigParser()
        config.read('../configuration.ini')

        f = open('dns.json')
        data = json.load(f)
        f.close()

        message = Crypt.encrypt(message)
        self.threadSignal.emit(message)

        found_domain = False

        start_time = time.time()

        # RUMORE
        request_number = random.randint(5, 10)
        for i in range(0, request_number):
            number_random = random.randint(0, len(data) - 1)
            fake_domain = data[number_random]["dominio"]
            ip = data[number_random]["ip"]

            answer = sr1(
                IP(dst=server) / UDP(sport=RandShort(), dport=53) / DNS(id=random.randint(0, 65535), rd=1,

                             qd=DNSQR(qname=fake_domain),

                             an=DNSRR(rrname=fake_domain,

                                      rdata=ip)), verbose=0)
            self.threadSignal.emit(repr(answer[DNS]))
            # time.sleep(random.randint(2, 10))

        while not found_domain:
            number_random = random.randint(0, len(data) - 1)
            fake_domain = data[number_random]["dominio"]
            ip = data[number_random]["ip"]

            if self.countConsonants(fake_domain)[0] % 2 == 0 and self.countConsonants(fake_domain)[1] >= 4:
                found_domain = True

        ttl = random.randint(2468, 10468)
        ttl_binary = bin(ttl)[2:].zfill(16)
        len_binary = bin(len(message))[2:].zfill(8)
        # pattern = random.randint(0, 15)
        # pattern_bin = bin(pattern)[2:].zfill(4)
        pattern_bin = config.get('CONFIG', 'pattern')

        binary = ''
        j = 0
        k = 0
        for i in range(0, len(ttl_binary)):
            if i % 2 != 0:
                binary += len_binary[j]
                j += 1
            elif i < 8 and i % 2 == 0:
                binary += pattern_bin[k]
                k += 1
            else:
                binary += ttl_binary[i]

        answer = sr1(
            IP(dst=server) / UDP(sport=RandShort(), dport=53) / DNS(id=random.randint(0, 65535), rd=1,

                         qd=DNSQR(qname=fake_domain),
```

```
                    an=DNSRR(ttl=int(binary, 2),
rrname=fake_domain,
                             rdata=ip)), verbose=0)
        self.threadSignal.emit(repr(answer[DNS]))
        # time.sleep(random.randint(2, 10))

        chunks = list(self.chunkstring(message, 16))
        for message in chunks:
            for i in range(0, len(message)):
                found_domain = False

                while not found_domain:
                    number_random = random.randint
(0, len(data) - 1)
                    fake_domain = data[number_random
]["dominio"]

                    if self.countConsonants(
fake_domain)[0] % 2 == 0 and self.
countConsonants(fake_domain)[1] >= 4:
                        found_domain = True

                dns_id = random.randint(0, 65535)
                binary_temp = bin(dns_id)[2:].zfill
(16)
                binary = ''
                message_binary = bin(ord(message[i])
)[2:].zfill(8)
                sequence_number = bin(i)[2:].zfill
(4)
                j = 0
                k = 0
                z = 0
                for i in range(0, len(binary_temp)):
                    if i % 2 == 0:
                        binary += message_binary[j]
                        j += 1
                    elif i < 8 and i % 2 != 0:
                        binary += sequence_number[k]
                        k += 1
                    else:
                        binary += pattern_bin[z]
                        z += 1

                new_dns_id = int(binary, 2)

                answer = sr1(
                    IP(dst=server) / UDP(sport=
RandShort(), dport=53) / DNS(id=new_dns_id, rd
=1,
                                 qd=DNSQR(qname=
fake_domain)), verbose=0)

                self.threadSignal.emit(repr(answer[
DNS]))
                # time.sleep(random.randint(2, 10))

                # RUMORE
                request_number = random.randint(1,
3)
                for i in range(0, request_number):
                    number_random = random.randint
(0, len(data) - 1)
                    fake_domain = data[number_random
]["dominio"]
                    answer = sr1(
                        IP(dst=server) / UDP(sport=
RandShort(), dport=53) / DNS(id=random.randint
(0, 65535), rd=1,
                                     qd=DNSQR(qname=
fake_domain)), verbose=0)
                    self.threadSignal.emit(repr(
answer[DNS]))
                    # time.sleep(random.randint(2,
10))

        end_time = (time.time() - start_time)
        self.threadSignal.emit("END in " + str(
end_time) + " seconds")
```

Listing 4. Send message Function

Come si può notare dal codice e come spiegato anche in precedenza, tra una query DNS contenente un'informazione segreta e un'altra sono inframezzate randomicamente delle queries DNS che non contengono alcuna informazione. Ciò è fatto solo per creare rumore e nascondere ancora ulteriormente le informazioni.
Inoltre, lato client, è stata sviluppata anche un'interfaccia grafica per permettere una migliore esperienza all'utente e facilitare l'uso dello strumento. L'interfaccia grafica è stata sviluppata tramite l'ausilio della libreria **PyQt5** e permette di scegliere il server DNS a cui mandare le queries e il messaggio da mandare, quest'ultimo può essere o scritto manualmente o selezionato da un file (.txt). Ciò è visibile in Figura 5

L'intero ambiente di sviluppo (sia client che server) è stato testato sulle piattaforme **MacOS** e **Linux**, in particolare su MacOS Big Sur versione 11.3 e Linux Lite 5.2 a 64 bit.

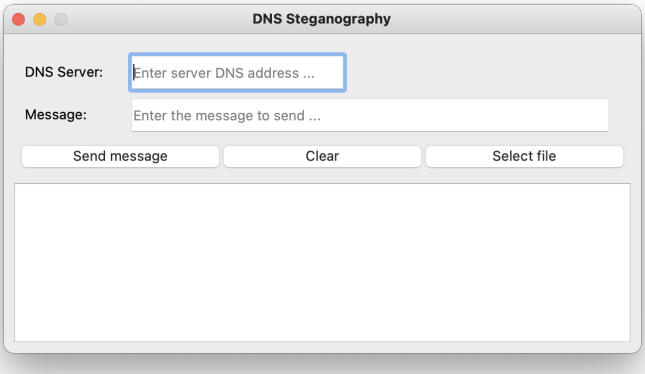

Fig. 5. Client Interface

V. Test e valutazione

Tutti i test sono stati effettuati in rete locale e tutte le queries DNS sono state monitorate tramite lo strumento **WireShark**. Come detto nei capitoli precedenti, tra una query e l'altra ho inframezzato delle query per creare rumore e i risultati sono riportati nella seguente tabella. In particolare, ciascun valore riportato è il frutto dell'overhead medio generato in un totale di 10 test. Ad esempio, per calcolare l'overhead generato dall'invio di un messaggio segreto di 8 bytes nel caso migliore (ovvero sono inframezzate 5 queries DNS contenenti il campo answer che non contengono alcuna informazione segreta e una

contenente informazioni di start session, e per ciascuna query DNS con il campo DNS ID modificato è inframezzata una sola query che non continene nessuna informazione) si è calcolato l'overhead generato da ciascun run e poi si è fatta una media aritmetica sui 10 run.

Overhead			
Bytes	Caso Migliore	Caso Medio	Caso Peggiore
8 Bytes	**3700 Bytes**	5700 Bytes	7000 Bytes
16 Bytes	6300 Bytes	10000 Bytes	12000 Bytes
32 Bytes	11200 Bytes	16300 Bytes	22200 Bytes
64 Bytes	21400 Bytes	31200 Bytes	**42200 Bytes**

Overhead

Nella seguente tabella è riportato l'overhead generato nella rete per poter trasemttere un messaggio segreto. Dalla tabella si può notare che per inviare un messaggio segreto di 8 bytes si è generato un traffico di 3700 bytes nel caso migliore, 5700 bytes nel caso medio e 7000 bytes nel caso peggiore.
Come si può notare, più il messaggio da inviare è grande maggiore è l'overhead generato sulla rete, infatti per trasmettere un messaggio lungo 64 bytes, nel caso peggiore, in totale si è generato un traffico di 42200 bytes. Ciò, però, è un traffico insolito per queries DNS, per questo motivo si possono inviare messaggi segreti lunghi al massimo 64 caratteri.

Infine, analizziamo il tempo di esecuzione per l'invio di un messaggio di diverse lunghezze.

Tempi	
Bytes	Secondi
8 Bytes	4,5 secondi
16 Bytes	12 secondi
32 Bytes	23 secondi
64 Bytes	50 secondi

Tempi

Ovviamente, i tempi di esecuzione dipendono dai tempi di risposta del server DNS di Google, però, in tabella sono riportati i tempi medi ottenuti da più esecuzioni del programma. Come potremmo aspettarci, maggiore è la lunghezza del messaggio da inviare, maggiore sono i tempi di esecuzione.

Inoltre, è da specificare che il traffico intercettato da WireShark non desta alcun sospetto, infatti sembrano query reali. Un esempio di query generata dal programma ed intercettata da WireShark è la seguente:

Come si può notare, non presenta nè campi alterati nè strani nomi di domini nè strane stringhe contenute nei campi TXT.

VI. Conclusioni

In questo progetto d'esame è stato presentato un nuovo meccanismo di "Covert Channel over DNS" che può essere

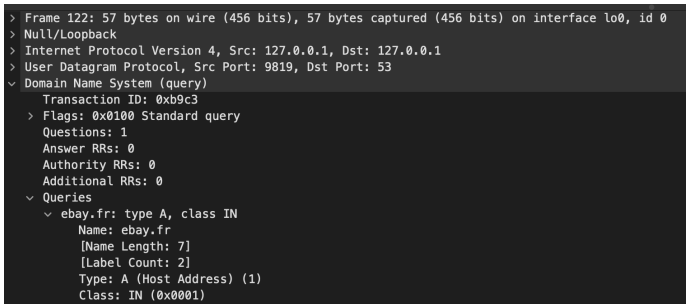

Fig. 6. WireShark

tranquillamente usato per nascondere informazioni segrete da inviare ad un server DNS compromesso. Come mostrato nel capitolo precedente, l'unico limite è quello di poter inviare messaggi di al massimo 64 bytes poichè, altrimenti, si genererebbero troppe queries DNS e ciò potrebbe destare dei sospetti e quindi essere sgamati.

References

[1] Michał Drzymała, Krzysztof Szczypiorski, and Marek Łukasz Urbański, "Network Steganography in the DNS Protocol", INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2016, VOL. 62, NO. 4, PP. 343-346, DOI: 10.1515/eletel-2016-0047.

[2] Abdulrahman H. Altalhi, Md Asri Ngadi, Syaril Nizam Omar and Zailani Mohamed Sidek, "DNS ID Covert Channel based on Lower Bound Steganography for Normal DNS ID Distribution", IIJCSI International Journal of Computer Science Issues, Vol. 8, Issue 6, No 3, November 2011.

[3] Christopher Hoffman, Daryl Johnson, Bo Yuan, Peter Lutz, "A Covert Channel in TTL Field of DNS Packets".

[4] Wojciech Mazurczyk1, Steffen Wendzel, Ignacio Azagra Villares and Krzysztof Szczypiorski, "On importance of steganographic cost for network steganography", DOI: 10.1002/sec.1085

[5] S.N Omar, I.Ahmedy, M.A Ngadi, "OIndirect DNS Covert Channel based on Name Reference for Minima Length Distribution", Proceedings of the 5th International Conference on 14 – 16 November 2011, IT & Multimedia at UNITEN (ICIMU 2011) Malaysia.