Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Indirizzi IPv6 generati tramite funzioni hash: Sperimentazione e Documentazione

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Davide Balestra

Sessione II - Secondo Appello
Anno Accademico 2019/2020

Introduzione

Durante il processo di risoluzione di un nome di dominio (DNS) l'overhead generato sulla rete spesso è di una portata non indifferente, inoltre la configurazione dei Name Server è tutt'altro che semplice: nel corso degli anni sono state sviluppate tecniche di load balancing o di caching al fine di ottimizzare le prestazioni di rete e semplificare la configurazione ed il mantenimento della coerenza dei record.
Lo studio svolto in questo elaborato si concentra nello sperimentare e documentare l'utilizzo di HashDNS: un server DNS virtuale capace di generare indirizzi IPv6 basato su VDE (Virtual Distributed Network), un progetto di virtualizzazione parte del gruppo Virtual Square.
L'idea di base di HashDNS è quella di sfruttuare l'ampiezza degli indirizzi IPv6 per risolvere un Domain Name utilizzando funzioni hash crittografiche: il calcolo e la generazione dell'indirizzo sono resi completamente automatici e la complessità della fase di configurazione ridotta.

L'intuizione alla base di HashDNS ha un potenziale che non si limita solamente all' automatizzazione del processo di risoluzione del nome ma sfocia nel concetto di IoTh (Internet of Threads) ovvero una visione di un modello di rete futuro in cui gli indirizzi IP non vengono assegnati alle macchine ma ai singoli processi (da non confonfere con Internet of Things).

L'obiettivo di questo elaborato è quello di creare una documentazione chiara al fine di incentivare e facilitare l'utilizzo dell'architettura da parte

degli utenti oltre che farne capire le potenzialità e suscitare l'interesse degli sviluppatori. Parallelamente alla documentazione verranno presentati risultati e considerazioni di una sperimentazione del sistema proponendo esempi concreti di utilizzo e fornendo spunti di miglioramento dell'attuale implementazione con lo scopo avere un codice più chiaro ed efficiente ed un utilizzo da parte dell'utente meno complicato possibile.

I test e la sperimentazione sull'utilizzo di HashDNS e delle architetture connesse sono stati eseguiti interamente su una rete sperimentale che si serviva di indirizzi pubblici sulla quale venivano connessi gli host utilizzando namespace virtuali di rete (vdens). L'analisi della struttura e della codifica dei pacchetti DNS è stata invece eseguita con l'ausilio del tool Wireshark.

L'elaborato verrà introdotto da una prima analisi della struttura dei pacchetti DNS seguita da un approfondimento sulle architetture utilizzate e la loro relativa configurazione al fine di evidenziarne i punti critici. Saranno poi ampiamente descritti dei casi d'uso dei sistemi in analisi in modo da introdurre la documentazione del codice attuale per poi concludere con le idee ed i modelli elaborati per lo sviluppo di una nuova implementazione.

Indice

Introduzione **1**

1 Struttura dei pacchetti DNS **7**

1.1 Struttura 8

1.2 Codifica 10

1.3 Codifica compressa 10

2 Hash DNS **13**

2.1 Architettura 14

2.2 Configurazione 15

2.3 Esecuzione 15

2.3.1 Main 16

2.3.2 Parsing 16

2.3.3 Tipologie di richieste 17

2.3.4 Generazione della risposta e terminazione 17

3 Fully Qualified Domain Name DHCP **19**

3.1 Utilizzo 19

3.2 Architettura 20

3.3 Esecuzione 21

3.3.1 Chiamata su un interfaccia di rete reale 21

3.3.2 Chiamata su una rete virtuale VDE 22

3.3.3 Parsing 22

4 OTIP - One Time IP 25
4.1 Architettura 25
4.1.1 Lato client 26
4.1.2 Utilizzo 27
4.2 Sicurezza 27

5 Parsing dei pacchetti DNS 29
5.1 Implementazione esistente 29
5.1.1 Skipname 30
5.1.2 Getname 30
5.1.3 Name2DNS 31
5.2 Analisi 31
5.3 Nuova implementazione 32
5.3.1 Esecuzione 33
5.3.2 Codifica 34
5.3.3 Schema 35
5.4 Considerazioni 36

6 Libreria per il parsing dei pacchetti DNS 39
6.1 Strutture dati 39
6.1.1 Schema 40
6.1.2 Inizializzazione 41
6.1.3 Inserimento di un elemento 42
6.1.4 Ricostruzione del nome 42
6.2 Integrazione 43
6.2.1 Gestione delle query multiple 43
6.2.2 Codice 44
6.3 Codifica 47
6.3.1 Costruzione della risposta 48
6.3.2 Query 48
6.3.3 Answer 50
6.3.4 Info authoritative e addizionali 50

6.4 Considerazioni 51

Conclusioni **53**

Bibliografia **55**

Ringraziamenti **57**

Capitolo 1

Struttura dei pacchetti DNS

Prima di affrontare in maniera più specifica lo studio delle architetture di HashDNS e degli altri progetti di Virtual Square è opportuno fornire delle basi più approfondite sul funzionamento del sistema di risoluzione di nomi di Dominio, più comunemente conosciuto come DNS.
In particolare, per comprendere il funzionamento dell'implementazione di HashDNS e le successive considerazioni fatte in merito ad una nuova implementazione è di fondamentale importanza conoscere come sono strutturati i pacchetti DNS. Durante la fase di sperimentazione questo tipo di ricerca è stata eseguita fornendosi di materiale online per la parte teorica e grazie all'ausilio di wireshark come tool di sperimentazione: sappiamo inizialmente che i pacchetti DNS transitano di default sulla porta 53 con protocollo UDP, il nostro filtro di ricerca sarà quindi "UDP.port==53".
Come per ogni protocollo di rete anche i pacchetti DNS hanno una loro struttura specifica ed utilizzano una codifica particolare per incapsulare i dati: sia per i pacchetti di tipo Query che quelli di tipo Answer abbiamo bisogno di un algoritmo che codifichi e decodifichi i nomi di dominio, in questo capitolo analizzeremo la struttura generale di un pacchetto DNS e come vengono organizzate le informazioni al suo interno approfondendo la codifica dei FQDN (Fully Qualified Domain Name) che fanno parte della Query/Response.

1.1 Struttura

Come per tutti i pacchetti di livello Applicazione, ovviamente anche quelli DNS saranno incapsulati all'interno dei pacchetti degli altri livelli dello stack di rete (Network, MAC, Transport Level): da una prima analisi risulta la seguente struttura:

```
▸ Frame 51: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0
▸ Ethernet II, Src: IntelCor_54:e7:6e (98:54:1b:54:e7:6e), Dst: Technico_c1:f2:4e (a4:91:b1:c1:f2:4e)
▸ Internet Protocol Version 4, Src: 192.168.1.88, Dst: 192.168.1.254
▸ User Datagram Protocol, Src Port: 36212, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x922f
  ▾ Flags: 0x0100 Standard query
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ www.google.it: type AAAA, class IN
        Name: www.google.it
        [Name Length: 13]
        [Label Count: 3]
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
    [Response In: 52]

0000  a4 91 b1 c1 f2 4e 98 54  1b 54 e7 6e 08 00 45 00   ·····N·T ·T·n··E·
0010  00 3b 91 a4 00 00 40 11  64 67 c0 a8 01 58 c0 a8   ·;····@· dg···X··
0020  01 fe 8d 74 00 35 00 27  16 77 92 2f 01 00 00 01   ···t·5·' ·w·/····
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   ·······w ww·googl
0040  65 02 69 74 00 00 1c 00  01                        e·it·····
```

Figura 1.1: Wireshark - struttura del pacchetto DNS

- Frame di livello fisico
- Frame di livello II (14 Byte)
- Livello di rete (Pacchetto IP, 20 Byte)
- Livello di trasporto, Pacchetto UDP (8 Bytes)
  - Porta sorgente (2 Bytes)
- Livello Applicazione, Pacchetto DNS (Lunghezza variabile)

Entrando nello specifico, la struttura del pacchetto DNS è la seguente:

- Header
  - Transaction ID (2Byte)
  - Flags (2Bytes)
  - Response (1Bit)
  - Opcode (4Bit)
  - Truncated (1Bit)
  - Recursion Desired (1Bit)
  - Reserved (1Bit)
  - Non Authenticated Data Accept/Deny (1Bit)
  - Questions (2 Byte)
  - Answer RRs (2 Byte)
  - Authority RRs (2 Byte)
  - Additional RRs (2 Byte)

- Queries (Lunghezza Variabile)
  - Name (Lunghezza Variabile)
  - Type (2Bytes)
  - Class (2Bytes)

  Nei pacchetti di risposta verranno inserite una copia delle query (una per ogni query) e le risposte vere e proprie in coda al pacchetto.

- Answer
  - Name: riferimento al CNAME (2Bytes)
  - Type (2Bytes)
  - Class (2Bytes)
  - TTL (4Bytes)
  - Length (2Bytes) = 4 Byte per IPv4
  - IP v4/v6 Address (4 Byte o 16 Byte a seconda della versione del protocollo IP utilizzata)

1.2 Codifica

Vediamo ora nello specifico come viene codificato un nome all'interno di un pacchetto DNS.
Ogni carattere è codificato in un byte esadecimale ed ogni stringa che compone il nome viene preceduta da un byte che identifica la sua lunghezza (la lunghezza massima di una label è 63).
Le stringhe sono quindi separate tra loro dal byte lunghezza della prossima stirnga e il nome viene terminato con un byte posto a 0.

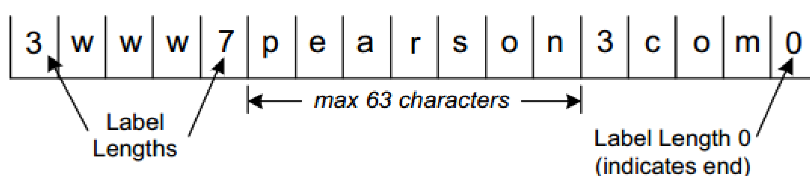

Figura 1.2: Codifca del FQDN nel pacchetto DNS

1.3 Codifica compressa

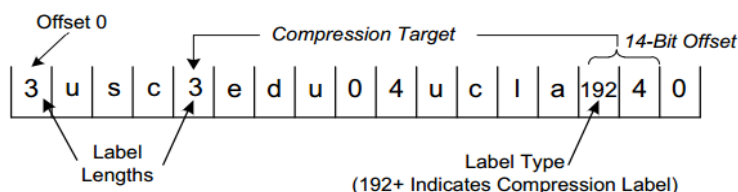

Figura 1.3: Codifica compatta

Le risposte DNS - come anche le query multiple - possono contenere informazioni duplicate all'interno dello stesso pacchetto, ad esempio un nome può comparire più volte o nomi diversi possono condividere delle stringhe.
In questi casi viene utilizzata una codifica compressa che permette di sfruttare stringhe già contenute nel pacchetto per comporre nuovi nomi evitando

ripetizioni, ottimizzando così la dimensione del paccchetto DNS.
Quando all'interno del pacchetto - nella sezione name - si ha un byte posto a 192 (0xc0 in esadecimale) vuol dire che la stringa che segue verrà codificata con lo schema compresso ed il byte successivo indicherà l'offset all'interno del pacchetto in cui si trova la stringa.
Come dallo schema in figura 1.3.

Capitolo 2

Hash DNS

Ora che abbiamo visto come sono strutturati i pacchetti DNS e più nello specifico come vengono codificate le stringhe che rappresentano i nomi di dominio al loro interno, siamo capaci di poter comprendere l'architettura di HashDNS e dei progetti di Virtual Square ad esso legati.
Hashdns nasce dall'intuizione di generare indirizzi IPv6 tramite l'utilizzo di funzioni hash: avendo un server DNS attivo su un dominio, quando si vuole risolvere un FQDN (Fully Qualified Domain Name) appartenente ad un sottodominio che utilizza HashDNS, la richiesta sarà gestita dal server DNS Virtuale (HashDNS) che agirà come fosse un proxy generando l'indirizzo IPv6 secondo il seguente schema:

- I primi 64 bit dell'indirizzo corrisponderanno all'indirizzo base del dominio (fornito dalla configurazione del Server DNS)

- La seconda parte dell'indirizzo sarà generata come hash del nome di dominio da risolvere.

In questo modo la configurazione diventa molto più semplice in quanto l'intero processo di risoluzione del nome è automatizzato, inoltre si limitano anche le operazioni di manutenzione dei record all'interno della struttura dei Server DNS diminuendo quindi i possibili problemi di coerenza tra i dati distribuiti.

2.1 Architettura

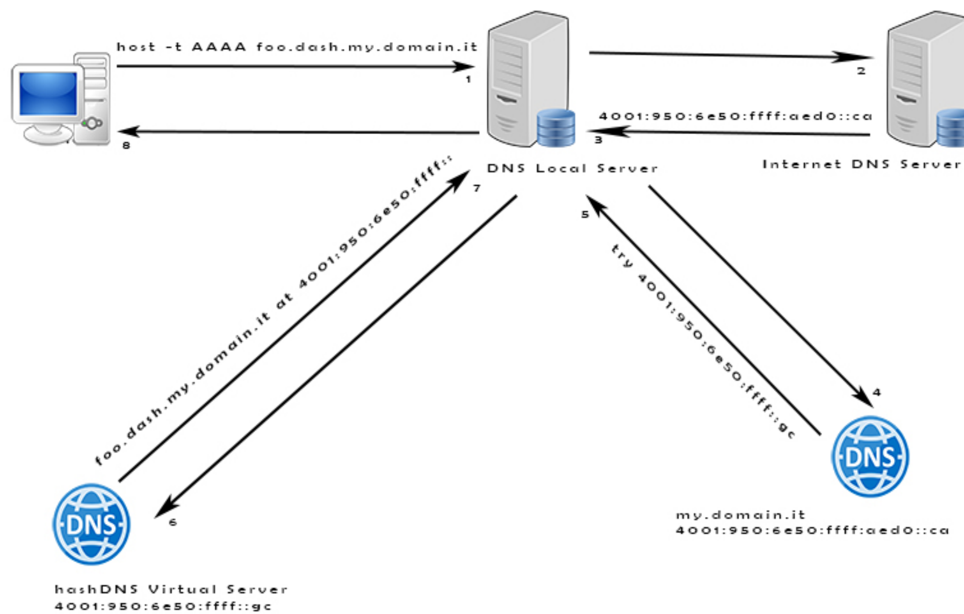

Figura 2.1: Architettura HashDNS

L'architettura di HashDNS consiste in

- Un server DNS Reale
- Server DNS Virtuale (HashDNS)
- DNS Client

Generalmente l'architettura viene configurata con i due Server DNS (Reale e Virtuale) in esecuzione sulla stessa rete ma ciò non è indispensabile: è sufficiente che i due server siano capaci di comunicare tra loro.
Quando il server DNS reale riceve una query da un client che richiede di

risolvere un nome appartenente ad un sottodominio che utilizza HashDNS, esso delegherà la risoluzione del nome al server Virtuale HashDNS.

2.2 Configurazione

Per far si che l'architettura descritta al paragrafo precedente funzioni è necessario fornire il server DNS reale di una configurazione particolare che preveda una delega ad HashDNS per i nomi del suo sottodominio ed un indirizzo base utilizzato per il calcolo degli indirizzi, ad esempio:

```
hash-dns         300 A 10.0.2.251
hash-dns         300 AAAA 4001:950:6e50:ffff::gc
hash             IN  NS   hash-dns
hash.my.domain.it IN  AAAA    4001:950:6e50:ffff::
```

File di configurazione del Server DNS (utilizzando bind9)

Questo vuol dire che:

- Risolvendo il FQDN hash.my.domain.it.my.domain.it si ottiene l'indirizzo base utilizzato per comporre gli indirizzi da HashDNS.
- Tutte le risoluzioni di nomi appartenenti a sottodomini di hash.my.domain.it verranno delegate all'indirizzo IP 10.0.2.251 (4001:950:6e50:ffff::gc per IPv6) ovvero gli indirizzi su cui dovrà essere posto in esecuzione HashDNS.

2.3 Esecuzione

Vediamo ora il funzionamento di HashDNS analizzando la sua esecuzione dalla chiamata alla risoluzione del nome di dominio soffermandoci ad analizzare gli aspetti implementativi più rilevanti.

2.3.1 Main

Quando HashDNS viene chiamato deve essere specificato un dominio di riferimento (che verrà utilizzato in seguito per risolvere l'indirizzo base) ciò può essere fatto tramite "mapdomain" o "mapfile" cioè rispettivamente specificando il dominio alla chiamata a seguito della clausola "-d" o utilizzando un file esterno in cui viene esplicitato il dominio indicando il path del file dopo la clausola "-f".
Prima di entrare nel corpo dell'esecuzione il main setta i parametri in accordo con i dettagli della chiamata: vengono chiamati i metodi setmapfile e setmapdomain (a seconda della chiamata) che appunto configureranno le variabili globali utilizzate dalle funzioni per il calcolo dell'indirizzo base.
Dopo la configurazione iniziale ed un check di tutti i parametri il main può configurare ed aprire il socket e poi chiamare il metodo main parse loop ovvero il principale loop di esecuzione di HashDNS che gestirà la ricezione e l'invio dei pacchetti sul socket.

2.3.2 Parsing

Dopo i controlli inziali del main e l'apertura del socket, il codice di HashDNS inizia la fase di parsing del pacchetto DNS: vengono inzialmente eseguiti dei test:

- Controllare che il pacchetto ricevuto sia conforme con la dimensione dell'header dei pacchetti DNS

- Controllare che il pacchetto DNS sia di tipo query

- Controllare che al suo interno ci sia esattamente una query

- Controllare che la query sia di tipo CLASS==IN

2.3.3 Tipologie di richieste

A questo punto abbiamo due possibilità:

- La query è di tipo AAAA o ANY: in questo caso viene chiamato il metodo computeaddr che seguirà i seguenti passaggi:
  - Cercare l'indirizzo base per il nostro dominio nel mapfile o eseguendo una DNS Query al mapdomain
  - Se nessuno dei due metodi riesce a ritornare un indirizzo base, viene generato un errore
  - Se l'indirizzo base esiste viene generato l'hash del fqdn e inserito in append all'indirizzo base così da comporre lintero indirizzo IPv6.
- La query è di tipo PTR (Reverse Query): viene chiamato il metodo getrevaddr che ha il compito di ottenere l'indirizzo IPv6 della query reverse:
  - HashDNS controlla nella sua cache la presenza di un record con IP corrispondente
  - Se esiste un record si ottiene un fqdn corrispondente da inserire nel pacchetto di risposta

2.3.4 Generazione della risposta e terminazione

In entrambi i casi, giunti a questo punto, sia se si tratta di una reverse query o di una tipologia AAAA/ANY il nostro server HashDNS ha generato l'indirizzo IPv6 o il fqdn da inserire nel pacchetto di risposta per il client: vengono quindi settati i parametri dell'header del pacchetto (flags) ed inserita poi la risposta prima di incapsularlo nei pacchetti di livello inferiore ed inviarlo sul socket.

Capitolo 3

Fully Qualified Domain Name DHCP

Il secondo progetto di Virtual Square che andremo ad analizzare è FQDN-DHCP (Fully Qualified Domain Name DHCP) ovvero un server DHCP per la configurazione e l'assegnamento automatico di indirizzi IPv6.
Quando il server FQDNDHCP riceve una query DHCP, controlla che essa contenga al suo interno un domain name (fqdn) per un record DNS di tipo AAAA (IPv6): se questo record esiste viene generato un IPv6 per il domain name e assegnato al client. In questo modo la configurazione degli indirizzi IP degli host di una rete diventa estramemente semplice, basta infatti fornire ad ogni host un domain name (fqdn) che accompagna le richieste DHCP in modo che il server virtuale possa generare autonomamente un indirizzo IPv6 da assegnare al client.

3.1 Utilizzo

L'intuizione dietro alla creazione di FQDNDHCP prevede che esso venga utilizzato in combinazione con un server DNS: si consiglia (ma non è indispensabile) di utilizzare il sistema con HashDNS configurando una rete di

server virtuali che forniscano servizi distribuiti basati sul modello di IoTh (Internet of Threads). Se si utilizza HashDNS per risolvere le DNS Query generate dal server DHCP, gli indirizzi IPv6 assegnati ai client saranno gli indirizzi calcolati tramite funzioni hash sulla base dei fqdn forniti dal client. FQDNDHCP + HashDNS diventa così un valido sistema di assegnazione automatica di indirizzi IPv6: i due server virtuali sono capaci di fornire una configurazione di rete agli host in modo completamente automatico.

3.2 Architettura

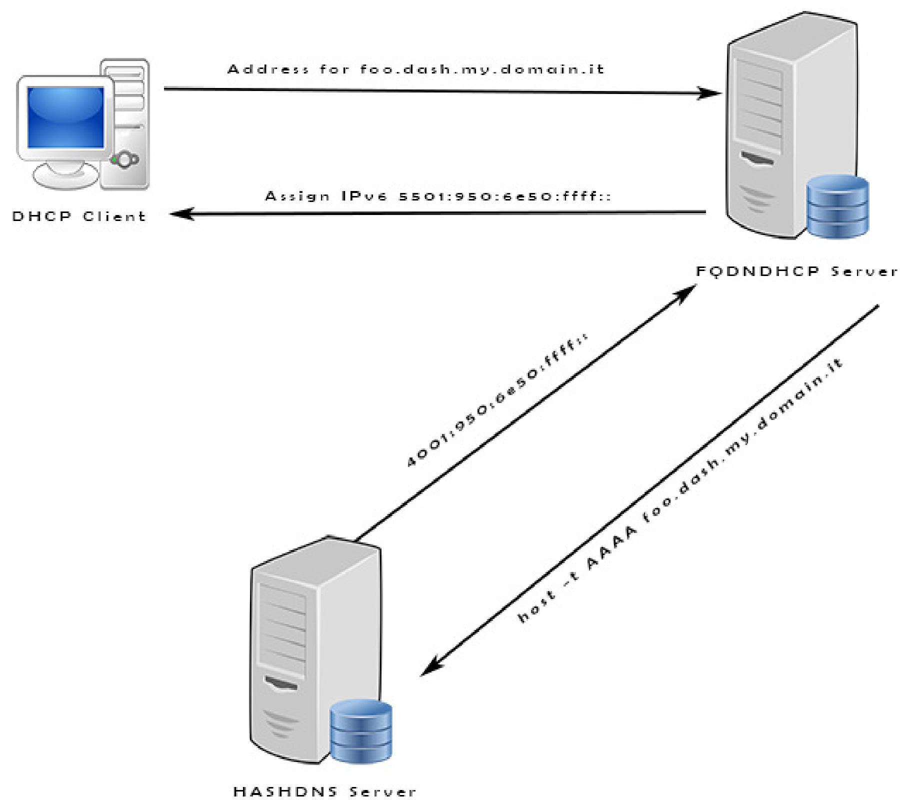

Figura 3.1: Architettura FQDNDHCP

L'architettura di FQDNDHCP consiste in:

- Client DHCP
- Server virtuale FQDNDHCP
- Server DNS (Generalmente HashDNS)

La query DHCP viene eseguita inserendo il "fully qualified domain name" per l'host: quando il server dhcp virutale riceve la richiesta dal client esegue una query DNS di tipo AAAA per ottenere un indirizzo IPv6 per il nome di dominio assegnato. Assumendo come nello schema in figura che il Server DNS utilizzato sia HashDNS, le query di tipo AAAA verranno gestite dal server virtuale che calcolerà un hash del nome di dominio fornito e genererà un indirizzo IPv6 che potrà essere assegnato al client.

3.3 Esecuzione

La chiamata a FQDNDHCP deve avvenire specificando l'interfaccia di rete su cui far girare il server virtuale o - alternativamente - una connessione virtuale VDE: il main poi sarà responsabile del settaggio dei parametri di chiamata.

3.3.1 Chiamata su un interfaccia di rete reale

In questo caso il primo step dell'esecuzione sarà quello di aprire l'interfaccia tramite il metodo open interface che restituirà un file descriptor da passare poi al loop principale che gestirà il proseguimento dell'esecuzione. Il metodo di apertura dell'interfaccia consiste in:

- Creazione del socket v6
- Settaggio dei parametri
- Binding

- Controlli

Il main loop crea e configura il socket v6, successivamente apre un buffer utilizzato per la scrittura del pacchetto di risposta e si mette in attesa di ricevere un pacchetto query. Quando il pacchetto arriva viene chiamato il metodo dhcpparse che è responsabile della decodifica del pacchetto in ingresso e della costruzione della risposta che verrà infine inviata sul socket aperto.

3.3.2 Chiamata su una rete virtuale VDE

Nel caso in cui invece la chiamata a FQDNDHCP viene fatta su un interfaccia virtuale VDE il primo passo è quello di generare un MAC Address per l'interfaccia virtuale (ciò viene fatto tramite il metodo generatemyaddr) dopodichè verrà aperta la connessione VDE. In questo caso il main loop che gestirà l'esecuzione si chiama main vde loop ed utilizza la primitiva vde recv per ricevere pacchetti sul canale VDE aperto. Una volta avvenuta la ricezione del pacchetto - dopo aver controllato che esso sia valido (non vuoto) e che gli indirizzi MAC corrispondono - viene chiamato il metodo vdepktin che gestisce l'estrazione del pacchetto di livello Applicazione decapsulandolo dai livelli inferiori.

3.3.3 Parsing

La fase di Parsing è comune ad entrambe le versioni. Viene estratto il fqdn (Fully Qualified Domain Name) dal pacchetto DHCP e successivamente eseguito un controllo sul tipo di richiesta DHCP. I due casi gestiti sono SOLICIT e REBIND appunto quando viene richiesto un nuovo indirizzo IP oppure quando viene richiesta una conferma per continuare ad utilizzarne uno già assegnato.

- Solicit: Nel caso viene chiesto un nuovo indirizzo IPv6, FQDNDHCP invia una richiesta DNS di tipo AAAA per il fqdn specificato. Se il server DNS non risponde viene generato un errore altrimenti viene

assegnato al client l'indirizzo IPv6 risolto dal DNS (HashDNS in questo caso).

- Rebind: In questo caso deve essere eseguito un ulteriore controllo (tramite il metodo check iana) per verificare che l'IP assegnato in precedenza per il nome di dominio specificato può essere confermato: in caso affermativo il client potrà continuare ad utilizzare il vecchio IP altrimenti viene inserito nel pacchetto di risposta un codice di errore.

Capitolo 4

OTIP - One Time IP

Il terzo progetto di Virtual Square che andremo ad analizzare prima di passare allo studio specifico del parsing dei pacchetti DNS è OTIP, ovvero One Time IP: un'architettura che riprendendo lo stesso concetto di "One Time Password" si propone di mettere a disposizione del client un server web che cambia dinamicamente il suo indirizzo IP allo scadere di un timer.
L'accesso al server da parte del client può avvenire solo utilizzando una password segreta condivisa tra i due: utilizzando OTIP quindi si va ad inserire un ulteriore livello di sicurezza alla comunicazione client/server, vedremo in questo capitolo più nello specifico la sua architettura ed il suo funzionamento..

4.1 Architettura

Come possiamo vedere dallo schema il funzionamento di OTIP si basa sull'utilizzo di VDE switch in una rete VDE dove è in esecuzione un web server (Otip Web Server virtuale) che calcola automaticamente il suo indirizzo IPv6 come risultato di una funzione hash sul suo nome di dominio, una password condivisa con il client ed il timer corrente. In questo modo sappiamo per certo che l'indirizzo IP è One Time, cioè verrà utilizzato una sola volta per un periodo di tempo limitato, dopodichè il timer cambierà e

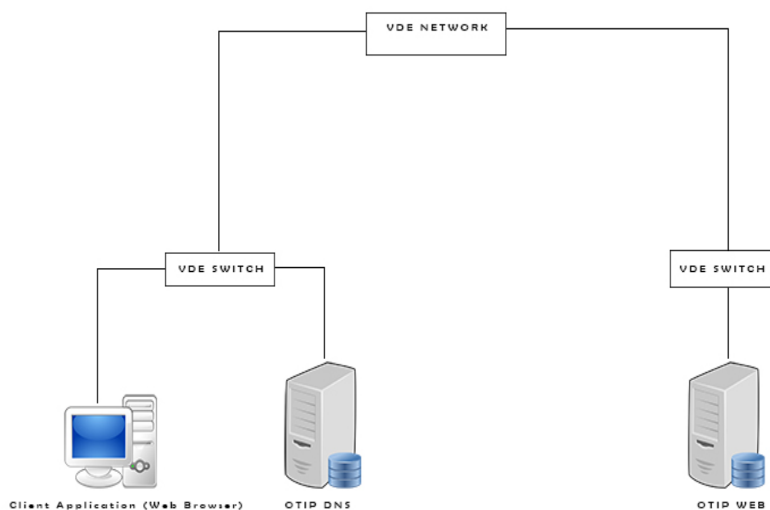

Figura 4.1: Architettura FQDNDHCP

di conseguenza l'indirizzo calcolato sarà diverso.

4.1.1 Lato client

Mentre il Server Virtuale Otip è in esecuzione dietro un vde switch, dal lato client l'architettura consiste in un host che effettua le richieste web ed un Server DNS Otip che risolverà i nomi di dominio richiesti dall'applicazione client.
Quando un' applicazione dal lato client (es. Web Browser) fa una richiesta per accedere al server web Otip, aggiunge in fondo al nome di dominio da raggiungere la password segreta, in questo modo la richiesta DNS conterrà anche la password ed il server DNS virtuale - sincronizzato con il timer del

web server - sarà in grado di calcolare esattamente lo stesso hash , fornendo così al client l'indirizzo IPv6 corretto.

4.1.2 Utilizzo

Come è già stato detto, per generare l'indirizzo IPv6 Otip necessita di:

- Password
- Timer
- Domain Name
- Indirizzo base

Quindi alla chiamata (sia del web server che del DNS o forwarder) è necessario fornire ad OTIP l'indirizzo base del suo dominio con il quale calcolerà tutti gli indirizzi IPv6 tramite funzioni hash. Un ulteriore parametro richiesto per l'esecuzione del web server è ovviamente la password segreta.

4.2 Sicurezza

Gli indirizzi generati da Otip sono One Time, ovvero possono essere utilizzati per un intervallo di tempo limitato prima di scadere. Oltre al questo primo livello di sicurezza che consiste nel cambiare automaticamente l'indirizzo IP su cui gira il web server, l'accesso da parte del client può avere successo solo se si conosce la password segreta.

Capitolo 5

Parsing dei pacchetti DNS

Come si può intuire dai capitoli precedenti, durante l'esecuzione di HashDNS - così come del dns server di Otip - la fase di ricezione del pacchetto DNS è seguita da una fase di elaborazione in cui viene costruita una risposta prima di poterla inviare sul socket. Necessariamente per elaborare una risposta occorre interpretare e decodificare i dati ricevuti dalla query per calcolare correttamente l'indirizzo IPv6: inziamo a parlare della fase di parsing.

5.1 Implementazione esistente

Una volta avvenuta la ricezione del pacchetto occorre ottenere il nome di dominio per poter calcolare l'indirizzo: il modulo che gestisce il parsing e la decodifica del pacchetto in ingresso è implementato dalla libreria name utils. I metodi che implementano le funzionalità di parsing sono:

- skipname(char *s).

- getname(char *buf, char *curin, char *name, char *limit).

- name2dns(char *name, char *out).

5.1.1 Skipname

Il metodo skipname ha lo scopo di restituire un puntatore alla posizione del pacchetto immediatamente successiva al termine del nome di dominio della query. L'implementazione segue uno schema di ricorsione, abbiamo tre casi:

- Il byte puntato dal buffer è posto a 0: ciò significa che la stringa che rappresenta il nome è terminata, possiamo quindi terminare l'esecuzione restituendo la posizione del buffer successiva a quella corrente.

- Il byte puntato è posto a 192: sappiamo quindi che il nome verrà rappresentato in forma compatta e che il prossimo byte rappresenta l'offset in cui trovare il nome. In questo caso è sufficiente restituire la posizione del buffer scorrendo di due byte (uno per il tag di codifica compatta ed uno per l'offset).

- Il caso ricorsivo avviene quando il byte corrente non è nullo nè posto a 192: dal capitolo 1 sappiamo che in questo caso il byte può rappresentare un carattere del nome o la lunghezza di una label che lo compone. Sappiamo però che ogni label deve essere preceduta dalla sua lunghezza quindi possiamo procedere con una chiamata ricorsiva scorrendo la posizione del buffer della lunghezza della label.

5.1.2 Getname

Questo metodo viene utilizzato per generare la stringa che rappresenta il nome di dominio partendo dal nome codificato nel pacchetto, la codifica può essere:
- Full version
- Compressa

Quindi il metodo necessita di un buffer che punta all'intero pacchetto DNS

per essere in grado di decodificare nomi compressi: se il buffer è nullo la compressione non è supportata.

5.1.3 Name2DNS

Questo metodo viene utilizzato nel caso in cui HashDNS deve risolvere una query di tipo reverse (PTR), name2dns trasforma quindi il fqdn ottenuto dalla cache locale nella versione codificata da inserire nel pacchetto di risposta. Ad esempio, se il fqdn è foo.dash.my.domain.it, allora la versione codificata sarà:

```
/3foo/4dash/2my/6domain/2it/0
```

Questo metodo viene chiamato durante l'esecuzione di dnsparse passando come secondo argomento un puntatore al payload del pacchetto di risposta in modo che la codifica del nome generata venga inserita automaticamente nella risposta DNS.

5.2 Analisi

L'implementazione esistente è funzionante ma non completa in quanto non gestisce alcuni casi particolari che possono verificarsi durante la normale esecuzione di un Server DNS.
In primo luogo non viene utilizzata nessuna struttura dati intermedia durante il processo di decodifica del nome contenuto nel pacchetto DNS al FQDN effettivo che viene poi utilizzato per il calcolo dell'indirizzo IP (o viceversa nel caso di reverse query). Questo implica che durante il parsing non vengono memorizzate le informazioni intermedie che potrebbero essere utili in un secondo momento per ricomporre la risposta utilizzando la codifica compatta: nell'attuale implementazione la codifica compatta viene utilizzata solamente per copiare il fqdn della query all'interno della risposta utilizzando il TAG standard (0cc0 = 192,12) che indica l'offset fisso in cui risiede il nome della query.

Questa modalità di codifica ci permette la sola gestione delle richieste e risposte singole: non siamo quindi in grado di gestire il parsing di pacchetti DNS contenenti query multiple.
Nasce quindi l'esigenza di trovare un nuovo metodo implementativo che preveda l'utilizzo di una struttura dati intermedia mantenuta in memoria durante il transito del pacchetto sul server, la quale potrà rivelarsi utile in fase di ricostruzione delle risposte (gestione delle query multiple) ma anche facilitare e semplificare il parsing migliorando le prestazioni e rendendo il codice più leggibile.

5.3 Nuova implementazione

L'idea è quella di implementare una nuova libreria che gestisca il parsing dei pacchetti DNS e nello specifico la decodifica dei nomi di dominio contenuti nei pacchetti: creando una struttura dati contenente le stringhe che compongono i nomi di dominio in fatti, saremo in grado - in fase di costruzione della risposta - di reperire le etichette che compongono i nomi in modo più semplice e diretto. L'intuizione di base è quella di utilizzare un albero come struttura dati intermedia per organizzare la codifica delle stringhe che compongono il fqdn. Occorre mantenere un albero im memoria durante l'esecuzione mentre il pacchetto DNS è in transito, sommariamente consideriamo il ciclo di vita dell'albero secondo questo schema:

1. Creazione dell'albero alla ricezione del pacchetto (Inizio Parsing)
2. Popolazione dell'albero (Parsing)
3. Creazione della risposta utilizzando l'albero
4. Deallocazione dell'albero al termine del loop (Una volta inviata Response)

5.3.1 Esecuzione

La libreria appena descritta dovrà gestire due fasi di esecuzione:

- Decodificare la stringa ed inserire il nome in una struttura dati
- Leggere la struttura dati e generare una stringa che rappresenti il fqdn sulla base del quale generare l'indirizzo IPv6 con funzioni hash.

Proviamo ora ad ipotizzare un algoritmo generico per la fase di parsing del pacchetto DNS (riformulando l'algoritmo presente nella vecchia implementazione):

- Ricezione del pacchetto sul socket e set up del buffer (inizio fase di parsing)
- Skip dell'header del pacchetto
- Controllare che il numero di query sia maggiore di 0, in caso contrario generare un errore (nella vecchia implementazione si controllava che il numero di query fosse esattamente uno, in questa nuova sperimentazione proviamo a fornire un servizio che gestisca tutti i casi possibili, compreso quello di query multiple).
- Salvataggio del numero di query ed entrata in un ciclo for (un'iterazione per ogni query)
- Saltare il nome ed ottenere un puntatore alla querytail (possiamo utilizzare il metodo skipname)
- Controllare la classe e la tipologia di query (occupiamoci ora solo di query di tipo ANY o AAAA, per quanto riguarda richieste reverse apriremo una sezione dedicata più avanti)
- Nel caso i controlli vengono passati, popolare la nostra struttura dati con il nome appartenente alla query
- Ottenere il fqdn della query corrente dalla nostra struttura dati

- Chiamare il metodo computeaddr sul nome di dominio appena ottenuto per calcolare l'indirizzo IPv6
- Costruire la risposta inserendovi l'indirizzo appena calcolato settando i parametri ed aggiungerla in coda al buffer
- Skippare la query corrente e posizionare il buffer sulla seguente (se esiste altrimenti si esce dal ciclo for) possiamo eseguire questa operazione facilmente avendo il buffer all'inizio della query tail e sapendo la sua dimensione fissa basta scorrere della dimensione della query tail
- ciclare al punto 5
- Settare parametri e flag del pacchetto
- exit

5.3.2 Codifica

Come visto nel capitolo riguardante la struttura dei pacchetti DNS, sappiamo che la codifica dei fqdn è la seguente

1. Versione Estesa 0x03|www|0x06|google|0x02|it|0x00
2. Versione compatta (DNS Response) 0xc0|0x0c

Ovviamente il nostro parser dovrà scandire tutta la stringa: il loop si fermerà quindi quando verrà incontrato lo 0.
La codifica compatta viene utilizzata nei pacchetti di response per abbreviare il domain richiesto nella risposta: avendo il pacchetto DNS una struttura fissa questo implica che il nome si troverà sempre nello stesso offset all'interno del pacchetto, cioè 12 (e così via nel caso di query multiple).
Nella costruzione della risposta basta quindi inserire i caratteri:
- c0 (192 per indicare la forma compatta)
- 0c (12 per l'offset).
Tuttavia quello di copiare il fqdn della query in testa alla risposta non è

l'unico utilizzo della codifica compatta nei pacchetti DNS, infatti possiamo ritrovare una compressione anche nella codifica di stringhe ripetute in diversi fqdn nel caso di pacchetti DNS con query multiple, questo perchè la codifica compatta non viene utilizzata solo per copiare il fqdn della query (TAG Standard c00c) ma anche per semplificare le ripetizioni di nomi simili all'interno dello stesso pacchetto.

5.3.3 Schema

Ipotizziamo che nel nostro pacchetto DNS Query siano presenti tre nomi di dominio codificati con lo schema descritto nel capitolo 1. Assumiamo ora che il pacchetto DNS query contenga queries multiple e che i nomi di dominio da risolvere abbiano delle stringhe in comune, ad esempio:

- Name = 3—foo—4—hash—2—my—6—domain—2—it—0
- Name = 3—bar—192—12
- Name = 6—subdom—4—dash—192—21

  Come possiamo vedere le etichette comuni ai nomi di dominio vengono rappresentate con codifica compatta.
  Prendendo come riferimento l'intuizione descritta al paragrafo precedente per la nuova implementazione, il processo di popolazione dell'albero di parsing avverrà nel seguente modo: il primo fqdn ad essere preso in analisi verrà decodificato stringa per stringa e contemporaneamente verranno inseriti i record [stringa, lunghezza] all'interno dell'albero (partendo da una radice) in modo che ogni stringa sia figlia dell'etichetta che la precede. Quando entreranno le etichette successive, la procedura di popolamento sarà la stessa: verrà inserito un nuovo figlio della radice che rappresenta la prima etichetta del nuovo nome di dominio e quando si incontrano dei riferimenti ad altre stringhe per la codifica compatta, il figlio dell'etichetta corrente verrà impostato

all'etichetta puntata dall'offset. Utilizzando questa procedura di popolamento dell'albero otterremo una struttura dati in cui - partendo dalla radice - sarà possibile visitare tutti i nodi fino alle foglie componendo i singoli fqdn che poi potranno essere utilizzati per il calcolo degli indirizzi IPv6.

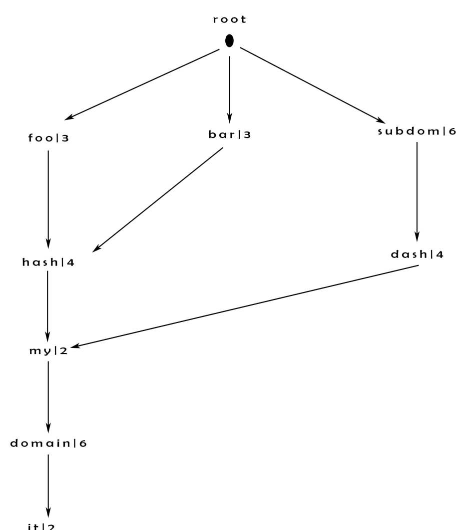

Figura 5.1: Schema dell'albero di parsing

5.4 Considerazioni

In questo capitolo abbiamo approfondito la fase di parsing dei pacchetti DNS che avviene prima del calcolo degli indirizzi IPv6 durante l'esecuzione del server HashDNS soffermandoci inzialmente ad analizzare l'implementazione attuale valutandone gli aspetti positivi ed i suoi limiti. Le limitazioni riscontrate sull'implementazione attuale ci hanno permesso di trarre nuove intuizioni e schemi di funzionamento utili allo studio di una nuova versione del sistema più efficiente ed in grado di gestire più casi generali possibili. Per

ora l'analisi condotta è rimasta sul piano astratto limitandosi a modellare le intuzioni e a fornire un primo schema di base: nel prossimo capitolo scenderemo ad un livello di dettaglio più basso andando a valutare e a discutere gli aspetti implementativi per la nuova versione di HashDNS.

Capitolo 6

Libreria per il parsing dei pacchetti DNS

Nel capitolo precedente ci siamo soffermati ad osservare come i pacchetti DNS vengono scansionati durante l'esecuzione di HashDNS per poi - dopo una serie di analisi - concludere esponendo i dettagli di una libreria per un parser più completo e performante avvalendosi di strutture dati aggiuntive. Abbiamo visto che una soluzione valida per la nuova implementazione del parser è quella di utilizzare un albero come struttura dati in cui inserire le etichette che compongono i nomi di dominio fornendo un primo schema generale della struttura dati ausiliaria ed una sequenza di azioni che descrivesse un primo algoritmo.
Passiamo ora ad un livello di dettaglio maggiore andando a vedere come le idee del capitolo precedente possono essere implementate.

6.1 Strutture dati

I nodi della struttura dati "Parsing Tree" sono caratterizzati da:

- Label (etichetta che compone il nome di dominio)
- Lunghezza dell' etichetta

I nodi del nostro albero saranno connessi tra loro in modo da formare una sequenza di etichette che andranno poi a comporre l'intero nome di dominio: una stringa verrà quindi inserita nell'albero di parsing come figlia dell'etichetta che la precede. Per quanto riguarda la codifica compatta, quando si incontra un byte posto a 192 il nodo di riferimento avrà come figlio il nodo puntato dall'offset, ad esempio:

- se abbiamo un fqdn codificato nel paccheto come "0x03|www|0x08|mydomain |0x03|org|0x00"
- verranno creati 3 nodi nell'albero
- org sarà figlio di "mydomain"
- mydomain sarà figlio di "www"

6.1.1 Schema

Per semplificare i metodi di aggiunta delle label e di ricostruzione dei nomi di dominio (che andremo ad approfondire più avanti in questo capitolo) è stato scelto di implementare la struttura albero secondo lo schema descritto dal seguente codice.

```
1 #ifndef PARSING_TREE_H
2 #define PARSING_TREE_H
3 #include <string.h>
4 #include <stdlib.h>
5 #include <utils.h>
6
7 typedef struct parsing_tree {
8   int length;
9   char *label;
10  parsing_tree *first_child;
11  parsing_tree *right_sibiling;
12 } parsing_tree;
13
14 //init the root node of parsing_tree
```

```
15 void init (parsing_tree *root);
16
17 //Decode and add next fqdn of DNS query to our data structure
18 void addName(char *buf, parsing_tree *head);
19
20 //Visit the parsing tree and generate a fqdn to fqdn char
     variable
21 void getNameFromTree(parsing_tree *head, char *fqdn);
22 #endif
```

In questo modo partendo da una radice, ogni nodo avrà collegato un solo figlio (first child) ed il suo fratello successivo (right sibiling) questo perchè nel processo di elaborazione degli indirizzi IPv6, essi verranno calcolati uno alla volta prendendo come fqdn di riferimento l'ultimo nome aggiunto all'albero. Utilizzando questa implementazione, per trovare l'utlimo nome aggiunto all'albero (in ordine cronologico) basta partire dalla radice, controllare se sono presenti dei figli e poi scorrere tutta la lista dei fratelli fino al termine; una volta raggiunto l'ultimo fratello (ultimo figlio della root inserito) basta visitare tutti i suoi figli fino a raggiungere una foglia, componendo così il nome di dominio.

6.1.2 Inizializzazione

Al momento della dichiarazione dell'albero di parsing, avremo inizialmente un puntatore alla radice dell'albero vuoto: occorre a questo punto inizializzare i parametri della radice correttamente, come possiamo vedere dal codice al paragrafo 6.2.2
Durante la fase di inserimento di un nome nella struttura dati viene creato un nodo per ogni etichetta che compone il nome: anche in questo caso il nodo viene inizializzato secondo lo schema del metodo init, e successivamente i suoi parametei verranno settati in modo opportuno.

6.1.3 Inserimento di un elemento

Come già descritto nei paragrafi precedenti, l'inserimento di un nome all'interno della struttura dati viene gestito dal metodo ricorsivo addName, in dettaglio:

- Se il carattere corrente è 0, termina
- Creare un nuovo nodo ed inserirlo come ultimo figlio della radice dell'albero
- Se il carattere corrente è 192, indica una codifica compatta, viene quindi configurato come figlio del nodo appena creato l'indirizzo del nodo descritto dall'offset.
- Se non si tratta di codifica compatta, vengono settati i parametri del nodo
- Si setta la lunghezza dell'etichetta
- Si copia l'etichetta carattere per carattere
- Chiamata ricorsiva sulla prossima etichetta

6.1.4 Ricostruzione del nome

Una volta inserito il nome nella struttura dati occorre visitare le etichette che lo compongono al fine di costruire una stringa contenente il nome di dominio da passare poi al metodo computeaddr che provvederà al calcolo dell'indirizzo IPv6: questa fase viene gestita dal metodo getNameFromTree. Il calcolo dell'indirizzo viene fatto per un nome alla volta (per tutti i nomi delle query) quindi di volta in volta il nome da prendere in considerazione sarà l'ultimo aggiunto all'albero in ordine cronologico. Per trovare l'ultimo nome aggiunto basta scandire la lista dei figli della radice fino ad arrivare all'ultimo fratello; una volta ottenuto il riferimento al primo nodo del nome da comporre basta visitare in sequenza tutti i suoi figli concatenando le etichette

separate dal punto, a questo punto l'esecuzione può procedere con il calcolo dell'indirizzo. Nel corpo del ciclo for appena creato avverranno le chiamate a addName e getNameFromTree che appunto gestiranno la popolazione dell'albero e l'estrazione del fqdn da passare al metodo computeaddr. Un'ultima modifica da apportare al vecchio codice per integrare l'utilizzo della libreria è quello di prevedere risposte multiple, infatti non sarà più sufficente costruire il pacchetto di risposta inserendo una sola answer ma occorrerà creare una singola risposta per ogni query risolta ed aggiungerla in coda al pacchetto prima di spedirlo sul socket.

6.2 Integrazione

Per garantire il corretto funzionamento della libreria descritta nel paragrafo precedente, è necessario apportare delle modifiche al vecchio codice di hashDNS per integrare l'utilizzo della libreria: in questa sezione vedremo quali sono i punti critici ed i cambiamenti da apportare al codice esistente.

6.2.1 Gestione delle query multiple

La vecchia implementazione prevedeva che il pacchetto DNS contenesse una sola query: l'utilizzo della nuova libreria parsing tree permette la gestione anche di richieste con query multiple quindi una prima modifica da apportare al vecchio codice consiste nel controllo del numero di query nel pacchetto che non saranno più limitate ad uno ma basterà controllare che il pacchetto presenti almeno una query. Ovviamente gestendo anche i casi di query multiple, a questo punto occorre prevedere un ciclo for (un'iterazione per ogni query) in modo da poter estrarre il fqdn di ogni richiesta e calcolarne l'indirizzo.

6.2.2 Codice

Vediamo ora la stesura del codice descritta dai paragrafi precedenti.

```
1 //Implementazione dei metodi della libreria "Parsing Tree"
2
3 void init (parsing_tree *root){
4   root->length=0;
5   root->label=NULL;
6   root->first_child=NULL;
7   root->right_sibiling=NULL;
8 }
9
10
11 void insertChild (parsing_tree *head, parsing_tree *node){
12   if (head->first_child == NULL)
13     head->firs_child=node;
14   else{
15     head=head->first_child
16     while (head->right_sibiling != NULL)
17       head=head->right_sibilng;
18     head->right_sibiling = node;
19   }
20 }
21
22
23 //Decode and add next fqdn of DNS query to our data structure
24 void addName(char *buf, parsing_tree *head){
25   if (*buf!=0){ //if 0 exit
26     //now create new node for our tree
27     parsing_tree node;
28     init(node);
29     insertChild(head,node); //insert node as child of head
30     if ((*buf & 0xc0) == 0xc0){ //compress mode
31       node->label = "192"
32       node->length = 0;
33       node->first_child = *buf+1;   //check
34     }
```

```
35     else{
36       node->length = *buf;
37       buf++
38       int nlength;
39       int copylength;
40       for (int i=0; i<node->length; i++){
41         //else copy str(buf) to head->label
42         nlength=strlen(node->label);    //get fqdn size
43         char ucopy[nlength];
44         snprintf(ucopy, nlength+1, "%s%s", node->label, *buf)
    ;
45         buf++;
46       }
47       node->label = ucopy;
48       addName(buf,node);
49     }
50
51   }
52 }
53
54
55 //Visit the parsing tree and generate a fqdn to char
56 void getNameFromTree(parsing_tree *head, char *fqdn){
57   if (head->first_child != NULL){ //if there is name saved in
      the tree
58     head = head-> first_child;
59     while (head->right_sibiling != NULL){ //find the last
    name inserted in the data structure
60       head = head->right_sibiling;
61     }
62     fqdn = NULL;  //clean buffer
63     fqdn=head->label  //first label in our fqdn
64     int nlength;
65     int copylength;
66     while (head->first_child != NULL){
67       head = head->first_child; //skip to next label
68       nlength=strlen(fqdn);   //get fqdn size
69       char ucopy[nlength];
```

```
70        snprintf(ucopy, nlength+1, "%s%s", fqdn, "."); //
    insert "." to get labels separate
71        copylength=strnlen(head->label);
72        snprintf(fqdn,nlength+1+copylength, "%s%s", ucopy, head
    ->label); //copy head->label in fqdn
73      }
74    }
75 }
```

Vediamo ora invece il codice di integrazione della libreria nel loop principale di HashDNS.

```
1 if (d16(dnspkt->nquery) < 1) goto dnsparse_err;
2   int x = d16(dnspkt->nquery);
3   int y =0;
4   for (int i=0; i<x;i++){
5     querytail = (struct querytail *) skipname(buf+sizeof(
    struct dnshead));
6     qtype=d16(querytail->type);
7     if (d16(querytail->class) != INCLASS)
8       goto dnsparse_err;
9     if (qtype == AAAATAG || qtype == ANYTAG) {
10      y++;
11      struct in6_addr addr;
12      struct reply *reply=(struct reply *)(querytail + 1);
13      addName(buf+sizeof(struct dnshead), root);
14      getNameFromTree(root, fqdn);
15      if(computeaddr (&addr, fqdn){
16        if (check_reverse_policy(&addr, &from->sin6_addr))
17          ra_add((char *)fqdn,&addr);
18        reply->shortname = e16(COPYNAME_TAG);
19        reply->type = e16(AAAATAG);
20        reply->class = e16(INCLASS);
21        reply->ttl = e32(1);
22        reply->len = e16(16);
23        memcpy(reply->payload,addr.s6_addr, 16);
24        n = ((char *)(reply + 1) + 16) - buf;
25        #ifdef PACKETDUMP
```

```
26          printf("REPLY\n");
27          packetdump(stderr, buf,n);
28        #endif
29      }
30    }
```

6.3 Codifica

Lo schema per la nuova versione descritto in questo capitolo permette di gestire il processo di decodifica del nome di dominio contenuto nel pacchetto DNS attraverso l'utilizzo di una struttura dati intermedia (albero di parsing). Come possiamo vedere dalla figura 6.1 durante il ciclo di esecuzione del codice di HashDNS, una volta ottenuto il nome di dominio dall'albero e calcolato l'indirizzo IPv6 occorre ricostruire un pacchetto di risposta valido da spedire sul socket. Nella versione attuale del codice questo processo viene eseguito ricopiando il pacchetto query e modificando opportunamente flags e parametri, inserendo poi in coda la risposta calcolata. L'obiettivo della libreria descritta in questo elaborato non si limita soltanto a fornire supporto in fase di decodifica delle informazioni ma si propone di dare supporto anche alla fase di codifica fornendo dei metodi opportuni che siano in grado di ricostruire automaticamente un pacchetto DNS di risposta senza dover andare a modificare manualmente i suoi parametri: vediamo in questo paragrafo come è possibile gestire tale fase.

Figura 6.1: Sequenza di esecuzione HashDNS

6.3.1 Costruzione della risposta

Come già approfondito ampiamente nel Capitolo 1, sappiamo che il pacchetto di risposta DNS è identico al pacchetto query a cui vengono aggiunte in coda le risposte ottenute con alcuni cambiamenti nei parametri dell'Header: possiamo quindi concludere che la prima fase della ricostruzione della risposta sia quella di recuperare i corretti parametri dal pacchetto query come Transaction ID e Flags (Occorre settare a 1 il flag opportuno per identificare il pacchetto come risposta, in più il parametro opcocde dovrà identificare che il pacchetto è una DNS Answer).
Oltre ai flags ed al transaction ID occorre configurare i campi "Numero di query" e "Numero di risposte" inserendo appunto il numero di query presenti nel pacchetto di richiesta ed il numero di risposte calcolate che saranno presenti nel pacchetto di risposta che stiamo costruendo.

6.3.2 Query

Il momento più delicato durante il processo di codifica e ricostruzione del pacchetto di risposta DNS avviene quando occorre ricostruire i fqdn codificati partendo dall'albero di parsing, per fare ciò apportiamo una piccola modifica alla struttura dati parsing_tree descritta nei paragrafi precedenti aggiungendo ad ogni nodo un nuovo parametro cioè il puntatore diretto al padre. In questo modo possiamo scorrere l'albero in due direzioni, nello specifico saremo in grado di partire da una foglia e risalire l'albero fino alla radice.

Il processo di codifica, avendo una stringa del tipo "foo.dash.mydomain.it" deve generare automaticamente in output:

"0x03|foo|0x04|dash|0x08|mydomain|0x02|it|0x00".

Vediamo uno schema di algoritmo:

- Ottenere la stringa del fqdn da codifcare (metodo getNameFromTree)

- Cominciare a scandire il nome al contrario separando le etichette
- Cercare una foglia dell'albero che abbia un'etichetta corrispondente a quella attuale nella scansione del nome (se non la si trova il nome non era stato inserito nell'albero)
- Partendo dalla foglia trovata risalire l'albero tramite il suo genitore controllando che l'etichetta corrisponda a quella attuale nella scansione della stringa
- Mentre l'albero viene risalito salvare in una coda separata i nodi scanditi
- Quando si arriva alla radice il loop può terminare
- Al termine della scansione percorrere la coda dei nodi creata al contrario inserendo in una nuova stringa prima la lunghezza della label e poi l'etichetta stessa
- Concludere la stringa con il carattere 0x00

Abbiamo così ottenuto una stinga codificata che rappresenta il nome di dominio da poter inserire nel pacchetto di risposta.
Utilizzando questo metodo vengono gestiti facilmente anche i casi di nomi "compatti" infatti se durante la scansione risalendo da un nodo al suo genitore ci si accorge che l'etichetta del nodo non corrisponde con quella della stringa in analisi allora sappiamo che la stringa che vogliamo codificare ha una codifica compatta.
In questo caso basta scorrere tutti i fratelli del nodo corrente fino a trovare quello che abbia l'etichetta che generi un match, a questo punto salviamo il puntatore al figlio (offset) e proseguiamo con la scansione risalendo l'albero a partire dal nodo trovato. Una volta terminata la scansione la ricostruzione del nome codificato avviene secondo lo schema descritto in precedenza ma una volta terminata la coda occorrerà inserire il carattere "0x0c" seguito dall'offset salvato in precedenza.

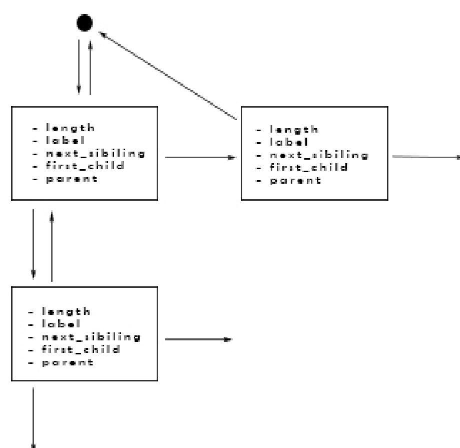

Figura 6.2: Albero di parsing con puntatori bidirezionali

6.3.3 Answer

La sezione "Answer" del pacchetto di risposta viene costruita codificando i nomi in modo compattto (in modo che gli offset facciano riferimento alla sezione "Query") e poi configurando i vari parametri, ad esempio:
- Type: necessariamente "AAAA/ANY" o "PTR"
- Class: "IN"
- TTL: fornito dal server
- Length: 16Byte (nel caso di Type="PTR" inserire la lunghezza del nome)
- Address: l'indirizzo appena calcolato dal metodo computeaddr

6.3.4 Info authoritative e addizionali

I parametri aggiuntivi da calcolare per costruire l'header della risposta corrispondono alle informazioni Authoritative e le info Addizionali sui Domain Name Servers.
Per quando riguarda il parametro "Numero di Authoritative Servers" possia-

mo configurarlo ad "1" in quanto si riferisce all'unico DNS Server del dominio principale, ovvero colui che effettua la delega al server virtuale HashDNS.
Per calcolare il Domain Name del server Authoritative basta troncare le prime due etichette della stringa del nome di dominio risolta nella query in modo da poter compilare la sezione "Autoritative Nameservers" del pacchetto inserendo nome di dominio, classe, tipologia, TTL e lunghezza.
Per quanto riguarda la sezione "Additional Information" basta eseguire una DNS query (di tipo ANY) al Server Authoritative per ottenere il suo/i suoi indirizzi IP. A questo punto configuriamo il parametro "Additional RRs" con il numero dei record ottenuti dalla query al server Authoritative ed i campi della sezione "Additional Information" con gli indirizzi IP appena risolti. Le sezioni Authoritative Nameservers e Additional Information vengono inserite in coda al pacchetto dopo la sezione "Answer".

6.4 Considerazioni

L'utilizzo della nuova libreria parsing tree integrata nel codice di hashDNS permette di gestire tutti i casi generali di pacchetti DNS, come ad esempio il caso di pacchetti con query multiple oltre che fornire un maggiore supporto durante il parsing e la decodifica dei nomi di dominio. La vecchia versione non disponeva di strutture dati ausiliarie in cui memorizzare le etichette dei nomi di dominio da risolvere, il risultato dell'integrazione della libreria è un maggiore supporto ed un codice più leggibile e modulare. L'implementazione descritta in questo capitolo è ancora in fase di test e sperimentazione, non è quindi da considerare applicabile in un contesto reale: è necessario eseguire test sul nuovo codice ed apportare le modifiche necessarie prima di passare ad una fase di pacchettizzazione ed ufficializzazione della nuova libreria.

Conclusioni

L'obbiettivo di questo elaborato è quello di fornire una documentazione dei sistemi virtuali del gruppo Virtual Square analizzandone il comportamento, l'esecuzione ed i vantaggi che possono portare al mondo reale. Non meno importante è anche cercare di attirare l'attenzione di utenti all'utilizzo dei sistemi o di ricercatori/sviluppatori alla contribuzione nella crescita del progetto.

Lo studio apportato nella struttura dei pacchetti DNS e del funzionamento del protocollo di risoluzione dei nomi di dominio è stata una fondamentale ricerca propedeutica alla comprensione del funzionamento dei sistemi stessi. Comprendere la struttura dei pacchetti ci ha permesso non solo di analizzare le attuali implementazioni dei sistemi virtuali ma anche di visualizzarle in maniera critica al fine di evidenziarne le criticità e fornire intuizioni per miglioramenti e nuove implementazioni.

Come già visto nei capitoli 2 e 3, i sistemi virtuali presi in analisi favoriscono il loro utilizzo in simbiosi in quanto permettono di costruire un environment di sistemi interconnessi che forniscano servizi automatizzati in modo da semplificare i processi di configurazione (nel caso di HashDNS e FQDNDHCP) ed apportare miglioramenti nel campo della sicurezza, come nel caso di OTIP.

I test eseguiti sulla rete sperimentale pubblica sono stati fondamentali in

primo luogo per stendere una documentazione chiara e sintetica ed in secondo luogo per comprendere quali fossero i casi specifici non ancora gestiti dall'attuale implementazione per poi passare quindi alla fase di studio della nuova versione. A questo punto siamo stati in grado di comprendere che l'utilizzo di una nuova libreria con strutture dati adeguate avrebbe portato miglioramenti al flusso di esecuzione, alla semplicità del codice ed alla "user experience" perciò il focus è stato spostato sull'implementazione di una struttura dati ad albero che permettesse l'inserimento e la decodifica automatica dei nomi di dominio contenuti nei pacchetti DNS.
L'albero di parsing, come visto nel capitolo 5, permette di semplificare il codice ed evitare ripetizioni durante la decodifica delle stringhe dei fqdn: l'utilizzo della libreria inoltre ha permesso ad HashDNS di poter gestire anche casi di query multiple (feature non ancora disponibile nella vecchia versione). L'elaborato è stato infine concluso fornendo uno schema di un algoritmo (partendo dalla vecchia versione) che integrasse la nuova libreria.

Il lavoro svolto ha permesso di produrre una documentazione dettagliata di un primo schema di base utile alla stesura di una nuova implementazione; il codice descritto nel Capitolo 6 non è quindi da considerare un'implementazione effettiva ma una base di partenza su cui effetuare test, analisi al fine di comprendere quali siano gli effettivi miglioramenti che la libreria apporterebbe al sistema prima di implementarne un codice definitivo.

Bibliografia

Fonti

DNS Packet Structure and Standards:
https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml
https://notes.shichao.io/tcpv1/ch11/

VDE: Virtual Distributed Ethernet
Renzo Davoli, 2005

IPv6 Hash-Based Addresses for Simple Network Deployment
Renzo Davoli, 2013

Internet of Threads
Renzo Davoli, 2013

OTIP: One Time IP Address
Renzo Davoli, 2013

Internet of Threads: Processes as Internet Nodes
Renzo Davoli, 2014

Virtual Square (V2) in Computer Science Education
Renzo Davoli Michael Goldweber, University of Bologna Xavier University

Virtual Square GitHub Repository
https://github.com/virtualsquare

Virtual Square Wiki (Old Version)
http://wiki.v2.cs.unibo.it/wiki/index.php?title=Main_Page

Virtual Square Official Wiki
http://wiki.virtualsquare.org/#!index.md

Ringraziamenti

Mi è doveroso dedicare questa sezione dell'elaborato alle persone che hanno contribuito non solo ad esso ma a tutto il percorso universitario vissuto in questi tre anni, senza le quali questa tesi non esisterebbe nemmeno.

In primis, un ringraziamento speciale va al mio relatore Renzo Davoli, per la sua pazienza e per i suoi indispensabili consigli che mi hanno accompagnato durante lo studio e la stesura dell'elaborato, senza i quali ciò non sarebbe stato possibile.

Ringrazio infinitamente i miei genitori e la mia famiglia che mi hanno sempre sostenuto durante il percorso di studi e non solo.

Un grazie di cuore va a tutti i miei colleghi con i quali ho condiviso l'intero percorso universitario, sia nei momenti più seri di studio, sia in quelli di divertimento necessari a rendere completa un'esperienza come questa.

Infine ma non meno importante, gli ultimi ringraziamenti vanno a tutti i miei amici, in particolare a tutti i membri di Alaska Records, ormai la mia seconda famiglia qui a Bologna, senza di voi non sarei mai riuscito in tutto questo.