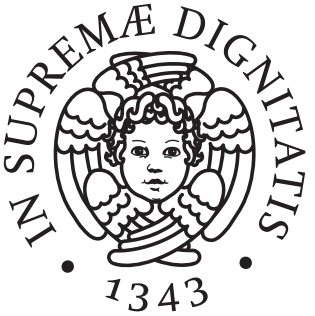


UNIVERSIT‘A DI PISA
DIPARTIMENTO DI INFORMATICA

Tesi di Laurea Triennale in

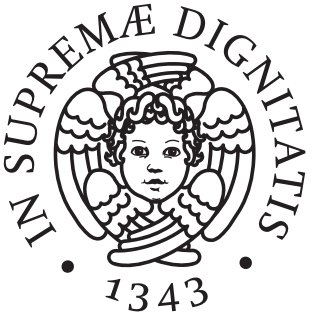
INFORMATICA

Titolo Tesi
Titolo Tesi

Relatore
Luca Deri

Candidato
Edoardo Maione

Anno Accademico 2017/2018



UNIVERSIT‘A DI PISA
DIPARTIMENTO DI INFORMATICA

Tesi di Laurea Triennale in

INFORMATICA

Titolo Tesi
Titolo Tesi

Relatore
Luca Deri

Candidato
Edoardo Maione

Anno Accademico 2017/2018

Dedica

Riassunto

Nelle Reti locali moderne, sia Domestiche che Aziendali, sta crescendo il numero di dispositivi ad alta capacitÃ di interazione con gli altri all'interno della stessa rete. A partire da stampanti di rete, proseguendo con i dispositivi mobili come Smartphone, Tablet, Wearable, ecc... che hanno avuto un aumento di diffusione esponenziale nell'ultimo decennio, fino ad arrivare agli elettrodomestici e alla domotica, anch'essa in crescita. In questo elaborato si vuole studiare e conoscere quali sono le informazioni che vengono scambiate tra questi dispositivi all'interno della rete locale per permettere un'interazione efficiente, ma soprattutto una ridotta, se non nulla, necessitÃ di intervento "specializzato" per quanto riguarda la configurazione di tali apparecchi.

Indice

Capitolo 1

Introduzione

Dispositivi e Applicazioni sono sempre più interconnessi tra loro, comunicando e scambiando informazioni, condividendo dati e interagendo per offrire servizi distribuiti ed autonomi. Basti pensare al semplice SSID di un Accesspoint WiFi, che si annuncia rendendosi visibile ai dispositivi in grado di connettersi, e diffondendo il proprio nome/ID del Router/Accesspoint; o ad una semplice stampante che si rende disponibile all'interno di una rete locale, identificandosi con il codice del Modello e diffondendo varie altre informazioni.

La necessità di rendere più autonoma possibile la comunicazione tra dispositivi e applicazioni, ha portato allo sviluppo di numerose tecniche e protocolli così detti di *Autoconfigurazione*, che permettono un setup autonomo del/dei dispositivo/i all'interno della rete locale, e quindi non necessitando di una configurazione "manuale".

Tutte queste interazioni sono rese possibili grazie la diffusione di informazioni, più o meno confidenziali, a seconda del protocollo utilizzato, e spesso compromettendo la privacy dell'utente, possessore del dispositivo o utilizzatore della specifica applicazione.

Inoltre possono essere presenti e ammesse all'interno della rete, tecnologie/protocolli che vanno a minare direttamente la sicurezza dell'intera rete, rendendo possibile modifiche nella configurazione di router o device di rete direttamente da remoto, come UPnP: il quale permette di aprire porte all'interno del router locale senza la necessità di autenticazione o permessi specifici.

Indubbiamente con tutte queste tecnologie si sono semplificate, se non addirittura rese completamente automatizzate, molte procedure di configurazione e interconnessione, rendendo accessibile a chiunque l'utilizzo di tali strumenti. Ma a quale prezzo? L'utilizzatore è a conoscenza di quali sono le informazioni scambiate all'interno della propria rete locale, e quali dati rende disponibile ad un eventuale ospite esterno/intruso nella propria Home-Network?

1.1 Traffico LAN Prima

Fino a qualche anno fa, all'interno delle nostre reti locali private, la quantità di traffico interno che vi transitava era pressoché nulla, dato che le uniche periferiche che avevano accesso alle rete erano i PC, e l'interazione fra di loro e le applicazioni era minima.

I primi a tentare un approccio di "Autoconfigurazione"e di *interconnessione automatica* furono gli sviluppatori Apple con il loro AppleTalk: in grado di mettere in comunicazione un gruppo di Macs all'interno di una rete locale LAN senza bisogno dell'intervento di alcun esperto, senza la necessità di alcun setup o di una struttura centrale che coordinasse o offrisse servizi per le periferiche, come un server DHCP o di un server DNS. Similmente, in seguito furono sviluppati NetBIOS e IPX, offrendo la medesima possibilità di interconnettere dispositivi che implementassero i suddetti protocolli.

Con l'avvento dello standard(tutt'oggi ancora NON definitivo) denominato ***Zeroconf***, nato dall'idea di AppleTalk, si sono susseguite numerose implementazioni e copiosi utilizzi del concetto di *autoconfigurazione* e 0 intervento esterno/strutture centrali per la configurazione e il coordinamento fra applicazioni/dispositivi. I primi a trarne vantaggio e trovarne subito un pratico utilizzo furono i costruttori di stampanti e, in generale, di dispositivi utilizzati in ufficio, non avendo avuto fino a quel momento la possibilità di includere interfacce utente per configurare manualmente le macchine, e quindi rendendo impossibile un agevole utilizzo di tali apparecchi all'interno della rete Aziendale/Domestica.

1.2 Cosa transita oggi all'interno delle NOSTRE Reti?

La quantità di informazioni che transita oggi all'interno della rete locale è veramente vasta, rendendo possibile l'utilizzo di dispositivi e servizi da essi offerti anche ad utenti non specializzati, completamente ignari di come sia resa possibile l'interazione; di contro, tutti questi dati, non solo riportano molte informazioni personali su dispositivi ed utenti che li utilizzano, ma inoltre per carpirle non è necessario compiere azioni specifiche, come introdursi all'interno del dispositivo, è sufficiente essere collegati alla stessa rete locale e recuperare tali informazioni dai pacchetti che vengono liberamente distribuiti all'interno della stessa, sia che il dispositivo sia realmente interessato che non. Questa situazione pone un'eventuale intruso/ospite nella rete, che è interessato a scoprirne la topologlia, in una condizione ottimale, limitandosi ad *ascoltare* le informazioni che gli vengono fornite degli altri dispositivi, senza intraprendere azioni di alcuna sorta nei confronti degli altri dispositivi, e quindi rendendone anche difficile l'individuazione.

1.3 Obiettivo

Lo scopo di questo Lavoro di Tesi è mostrare le vulnerabilità delle reti locali in termini di dati sensibili e privati, cercando di acquisire il maggior numero di informazioni possibili riguardo i nodi della rete, in modo completamente passivo, e identificando quali dispositivi sono connessi attualmente alla Rete Locale a cui abbiamo accesso. Questo mette in evidenza quante e quali informazioni vengono scambiate all'interno della rete, rendendo consapevoli gli utilizzatori di tale rete, di quali saranno le informazioni private che verranno diffuse tramite i loro dispositivi ad essa collegati. Grazie a tale consapevolezza, chi è addetto alla gestione e progettazione della rete può decidere eventualmente di separare il traffico in sotto-reti isolate, in modo tale da arginare eventuali diffusioni di informazioni sensibili, pur mantenendo e usufruendo di tutti i vantaggi che una comunicazione Broadcast/Multicast fra dispositivi in una rete locale comporta: per esempio auto-configurazione e scambio rapido di dati all'interno della rete locale, relegandolo nella propria LAN.

Al fine di raggiungere tale scopo, è stato approntato uno studio sulla metodologia di raccolta di tali informazioni, e la loro organizzazione ed elaborazione, identificando la natura dei dispositivi che popolano una generica rete locale, i servizi da loro offerti, e in alcuni casi, i rapporti/connessioni che hanno fra loro.

Come risultato di tale studio, è stato implementato uno strumento per l'analisi automatica di una rete locale, in grado di fornire informazioni più o meno dettagliate riguardo la topologia della rete, alla quale si ha libero accesso, e quindi identificando tutti quei dispositivi che annunciano e offrono servizi al suo interno: Dispositivi Mobili, Stampanti, Workstation di vario genere e Media-devices.

1.4 Struttura della tesi

Riassumere sommariamente ogni capitolo.

Capitolo 2 Panoramica dello studio approntato e del tool sviluppato per questo lavoro di Tesi.

Capitolo 3 Quali sono gli altri studi a riguardo.

Capitolo 4 Descrizione dettagliata degli strumenti usati e sull'implementazione.

Capitolo 5 Conclusioni tratte da questo studio e spunti per lavori futuri.

Appendice A Scrivere il riassunto.

Appendice B Scrivere il riassunto.

Capitolo 2

Lavoro: Analisi mDNS e DB-lsp-DISC

Il lavoro di tesi è partito da una raccolta di tutti quei protocolli di rete che utilizzassero lo scambio di pacchetti Multicast e Broadcast locale. ... Un primo approccio è stato il documentarsi riguardo gli indirizzi ufficialmente registrati per le comunicazioni Multicast LAN, e come risultato ho scoperto alcuni protocolli interessanti ... sono un'infinità! La ricerca mi ha portato alla scoperta di vari protocolli, alcuni simili tra loro, ... Il primo incontrato ...

Difronte ad una quantità enorme di protocolli, ho optato per un'approccio più pratico per verificare quali di tutti quei protocolli generassero pacchetti nelle nostre reti casalinghe o aziendali, catturando il traffico con uno strumento Open-Source chiamato Wireshark[1]. Ho quindi iniziato ad effettuare catture di pacchetti in svariate reti alle quali ho abituale accesso, filtrando il traffico ottenuto, tenendo quindi solo tutti quei pacchetti aventi come indirizzo di destinazione un indirizzo Broadcast (255.255.255.255) o Multicast (in range 224.0.0.0 fino a 239.255.255.255). Un'ulteriore scrematura preliminare dei pacchetti catturati è stata quella per escludere tutti quei pacchetti che venissero utilizzati da protocolli di rete adibiti alla configurazione e gestione della rete stessa, come per esempio i pacchetti ARP, ICMPv6, DHCPv6, e simili, quindi non contenenti informazioni rilevanti sulla natura dei dispositivi che li diffondono.

[1] descritto in seguito

2.1 Protocolli Multicast/Broadcast

All'interno del traffico restante, ho iniziato a raccogliere informazioni riguardo tutti i protocolli di livello applicativo restanti, valutando se le informazioni che contenevano potessero essere rilevanti ai nostri scopi.

2.1.1 Link-local Multicast Name Resolution

LLMNR è un protocollo che permette la risoluzione di indirizzi Ipv4 o IPv6 a partire dai nomi locali senza la necessità di un'entità centrale come un server DNS. Ci sono svariati altri protocolli che svolgono la medesima vunzione di risoluzione nomi DNS, e questo è stato pensato per sostituire l'entità centrale, e supportare anche l'utilizzo di IPv6. Questo protocollo non si è rivelato utile allo scopo di rivelare la natura o informazioni utili sui dispositivi nella rete, perché nelle catture effettuate sono risultati solo pacchetti contenenti query, e quindi richieste per la risoluzione di nomi per ottenere l'indirizzo IP, senza la risposta a conferma che quell'effettivo host sia collegato in quel momento alla rete. Questo è dovuto dal funzionamento dello stesso protocollo: un'host manda in multicast la richiesta per risolvere un nome, se un'altro host in ascolto è ?autoritativo? per quel nome, inoltra la richiesta in unicast direttamente a chi ha fatto domanda di risoluzione; quindi, per lo strumento utilizzato per la cattura del traffico, non è possibile reperire pacchetti che non siano direttamente indirizzati alla macchina sulla quale si sta facendo girare Wireshark, oppure traffico multicast/Broadcast.

2.1.2 NetBIOS-NS

Name Service(NS) è un servizio del protocollo NetBIOS[3], ideato da IBM e Sytec per la PC-Network[2] all'inizio degli anni Ottanta, e che con l'avvento delle reti standard è stato adattato(ma non abbandonato) per lavorare su altri protocolli, come TCP/IP[3]. NBNS è un'altro servizio che si occupa della registrazione e risoluzione dei Nomi nelle reti locali, rientra tra i primi servizi distribuiti atti a svolgere tale compito. Il suo funzionamento si divide in 2 fasi: Registrazione, nella quale un nuovo nodo si registra con un nome unico all'interno della rete, verificando prima che non vi sia quindi un'altro host già registrato con lo stesso nome, e Risoluzione, con la quale un nodo della rete richiede un indirizzo IP a partire da un nome simbolico locale. Studiando le informazioni contenute nei pacchetti catturati, oltre alle solite querys per la risoluzione dei nomi, si trovano anche pacchetti contenenti

[2]Tipo di rete locale

[3]chiamato anche NBT o NetBT

le richieste di registrazione dei vari nodi appena collegati, che li identificano univocamente all'interno della rete locale, e quindi fornendo un'utile(in alcuni casi) informazione riguardo la macchina: ovvero un'host che ha al suoi interno il protocollo NetBIOS in funzione.
Pur essendo molto interessante, NBNS non è stato usato in questo lavoro di tesi, sia per mancanza di tempo in relazione alle informazioni fornite, sia perché è stato identificato un'altro protocollo che offre ulteriori dettagli sui dispositivi che lo utilizzano, e anche i nomi degli stessi dispositivi.

2.1.3 Microsoft Windows Browser Protocol

Questo è un protocollo per la scoperta dei servizi offerti all'interno della rete/sotto-rete locale, ideato per i sistemi operativi Microsoft, il quale permette di gestire ed usufruire di tali servizi(condivisione di file, stampanti, ed altro...). In pratica, tramite una organizzazione di nodi gerarchica, permette di tenere traccia dell'elenco completo dei servizi presenti e diffondere tali informazioni ai nodi connessi alla sotto-rete locale, il tutto autogestendo l'assegnazione dei ruoli per la registrazione e assegnazione dei vari compiti necessari per il funzionamento del protocollo.
Il protocollo è gestito tramite una struttura gerarchica di nodi[7], ognuno dei quali svolge un determinato compito, e offre/riceve servizi al/ai nodi di ?grado? superiore e inferiore. Il nodo radice di tale struttura viene chiamato *Domain master browser*(o anche *Primary Domain Controllers: PDCs*), ed è responsabile della gestione delle liste di tutti i servers, una per ogni sotto-rete del dominio con all'interno un nodo *Master browser*. Al di sotto del PDCs, uno per ogni sotto-rete, si trovano i *Master browsers*, i quali si occupano di gestire le ?browse lists? del loro sotto-dominio di competenza ed inoltrarle al PDCs di sopra, e ai *Backup Browsers* al di sotto. Proseguendo nella gerarchia, al di sotto troviamo appunto i *Backup Browsers*, che diffondono individualmente a computers che ne fanno richiesta le informazioni raccolte nelle liste dei Master Browsers. Se si rendesse necessario, ci sono dei nodi che vengono etichettati come *Potential Browsers*, pronti a sostituire un eventuale Browser non più funzionante. In fine ci sono il resto dei nodi della rete chiamati *Nonbrowsers*, che sono appunto il resto delle macchine che non sono in grado di diffondere o tenere traccia delle liste di Browsing, ma che fanno parte della rete e offrono/richiedono servizi.
Dopo aver verificato quali informazioni sono contenute nei pacchetti catturati, si sono denotati vari tipi di messaggi, tra cui: *Browser Election request*, con i quali il protocollo si autogestisce, eleggendo il nodo più consono al compito da svolgere; *Get Backup List Request*; *'Local-Master'/Host/Request/'Domain-Workgroup' Announcement*, con i quali si rendono pubbliche indicazioni su

come raggiungere le varie macchine e quali servizi offrono. Ci sono vari altri tipi di messaggi per questo protocollo, ma non ne sono stati catturati. Le informazioni che se ne possono ricavare dagli *Announcement* sono decisamente rilevanti, identificando il nodo che le annuncia con un nome, che tipologia di macchina sia(workstation, server, ...), il produttore, ... e altri dettagli.

2.1.4 Multicast DNS-(Service Discovery)

mDNS[4] è un protocollo che si pone a sostituzione di un normale DNS centrale, dove magari in una piccola rete non vi è la possibilità/necessità di averne uno. In pratica si occupa di risolvere i nomi locali, con estensione ".local", tramite una richiesta da parte dell'host ad un indirizzo multicast(IPv4 224.0.0.251 / IPv6 ff02::fb), porta UDP 5353, inviando un messaggio dello stesso formato delle query DNS, ed ottenendo risposta, sempre in multicast, da un qualsiasi dispositivo(solitamente chi possiede il nome richiesto) che conosce l'indirizzo IP corrispondente.
Avendo la stessa struttura del protocollo DNS, oltre che alla risoluzione dei nomi, mDNS implementa anche il meccanismo DNS-Service Discovery[1], il quale permette la scoperta di istanze con nome di servizi nella rete locale, usando le querys standard DNS. Ogni servizio è identificato tramite un nome composto in notazione 'puntata', conforme al meccanismo gerarchico di nomi DNS, il quale è così suddiviso: *Istanza.Servizio.Dominio*. *Istanza* identifica univocamente il particolare dispositivo che offre il relativo *Servizio*, che a sua volta identifica il Tipo specifico di servizio offerto e il protocollo usato, ed in fine il *Dominio* riporta lo specifico dominio all'interno del quale il servizio è offerto (nel caso di mDNS è sempre '.local'). Per quanto riguarda i vari Tipi di servizi offerti in rete, IANA fornisce una lista[9] dei servizi ad oggi registrati, ed offre la possibilità di registrarne di propri, ma è comunque possibile utilizzare anche tipi non registrati e proprietari, senza la necessità di registrarne il nome.
Ogni dispositivo che vuole condividere/offrire un servizio, annuncia in multicast mDNS i record DNS-SRV e DNS-TXT. I record SRV riportano il nome, come descritto in precedenza, *Istanza.Servizio.Dominio*,e l'host con porta di destinazione a cui fare riferimento per richiedere il servizio. I record TXT sono opzionali, quindi non tutti i servizi annunciati li diffondono e non tutti i servizi dello stesso tipo rendono noti gli stessi campi, riportano alcuni dettagli del servizio offerto, e molto spesso, anche informazioni riguardo il dispositivo che si rende disponibile ad offrire il servizio.
Il dispositivo che invece vuole fare richiesta, usufruire, o semplicemente scoprire se il un dato servizio è reperibile, acquisisce la lista dei servizi disponibili effettuando delle query in multicast per ottenere record DNS-PTR, e richie-

dendo il nome del servizio del tipo: *Servizio.Dominio*. Come risposta, se nella rete è presente un host che ha le informazioni richieste, viene inoltrato in multicast un messaggio mDNS contenente, oltre al PTR che indica il nome completo dell'istanza del servizio, vengono forniti anche i record REV ed eventuali TXT collegati citati in precedenza.
Fra tutti i vari protocolli di Multicast/Broadcast rilevati, mDNS ha catturato maggiormente l'attenzione, sia per quanto riguarda la quantità di traffico generata, ovvero le molteplici query di discovery per i servizi presenti; sia per le informazioni specifiche contenute all'interno, che talvolta ha reso possibile di identificare, con una buona dose di affidabilità, il tipo e modello dispositivo fisico con tanto di dettagli tecnici riguardanti i pezzi hardware che lo compongono, il software che in quel momento è in esecuzione, e molto altro.

2.1.5 Dropbox LAN sync Discovery Protocol

Dropbox, una delle applicazioni più usate per il servizio di cloud storage e file sharing tramite Internet, utilizza per la sua versione desktop un protocollo chiamato Dropbox LAN sync Discovery Protocol(oppure in breve db-lsp-disc), con il quale incrementa la velocità di sincronizzazione[8] tra gli host che condividono le stesse cartelle all'interno della medesima rete locale. Questo inoltre evita anche che per ogni cartella condivisa, avvenga la sincronizzazione fra l'host che la condivide e i server di Dropbox, generando del traffico superfluo all'esterno della rete locale, e quindi limitando lo scambio di dati all'interno della LAN.
Questo meccanismo è reso possibile tramite lo scambio dei pacchetti db-lsp-disc fra gli host Dropbox, all'interno dei quali vi sono varie informazioni, tra cui: un identificatore unico, generato al momento dell'installazione, che identifica l'host; alcuni campi di utilità come versione app, display name, e porta usata per lo scambio di dati; ed infine il campo più interessante, namespaces, il quale riporta l'elenco di cartelle condivise in Dropbox con altri utenti. In realtà, Namespaces non riporta effettivamente l'elenco dei nomi delle cartelle, ma un'elenco di id unici che identificano le varie cartelle condivise. Questa è un'informazione molto preziosa, dato che pur non conoscendo nulla di un nodo della rete, se ne può rivelare le interazioni con altri dispositivi ad essa collegati, e quindi carpire le interazioni fra gli utilizzatori della rete locale.

2.2 Panoramica del Lavoro

Relativamente a questo lavoro di tesi, questi 2 ultimi protocolli si sono rivelati molto interessanti, fornendo informazioni decisamente 'riservate' e private,

ma soprattutto troppo facilmente accessibili, essendo recapitate direttamente a qualsiasi dispositivo collegato alla LAN, interessato o meno, il quale si ritrova in modo completamente passivo tali dettagli.
Partendo da questa considerazione, si è deciso di sviluppare un piccolo componente software che analizzi in automatico tutte queste informazioni e le raccolga in una struttura dati. Questa struttura dati racchiuderà ogni dettaglio dei relativi nodi utilizzatori di tali protocolli, rivelandone la natura, le 'abitudini', gli applicativi software in esecuzione, e in alcuni casi anche il possessore del dispositivo e le interazioni con altri utenti/dispositivi connessi alla medesima rete locale.
Per la raccolta del traffico dati è stato utilizzato, come per lo studio del traffico della rete locale, l'applicazione Wireshark(e la relativa versione in-line T-Shark), rivelatosi lo strumento perfetto a tale scopo, dato che permette di catturare tutto il traffico che transita dalla scheda di rete del dispositivo sul quale è in esecuzione. Il traffico poi è stato filtrato eliminando tutti i pacchetti unicast, ottenendo quindi una cattura di soli pacchetti Broadcast/Multicast.
Per la realizzazione del componente software, scritto in Python, è stato utilizzato un wrapper di T-Shark, anch'esso scritto in Python, il quale permette di leggere i file di cattura precedentemente ottenuti, e accedere ai singoli pacchetti del file e quindi reperire le informazioni contenute all'interno dei singoli campi.

2.2.1 Funzionamento

Di seguito verrà descritto il funzionamento dell'algoritmo che compone il software sviluppato per questo lavoro, tralasciando i dettagli tecnici nel prossimo capitolo, e limitandone la descrizione ad una panoramica completa.
La struttura principale è composta da un insieme di dispositivi, identificati univocamente dal loro MAC-Address, ognuno dei quali contiene: gli ultimi indirizzi IPv4 e IPv6 conosciuti(i quali ovviamente potrebbero essere cambiati da un'assegnamento dinamico), l'ipotetico possessore del dispositivo, la tipologia di dispositivo supposta a seguito dell'analisi, un elenco di 'alias' con i quali viene identificato il dispositivo, un elenco di tutti i servizi offerti dal dispositivo, rilevati tramite le informazioni contenute all'interno dei pacchetti mDNS, ed infine un campo che riporta tutte le informazioni raccolte dal protocollo Dropbox db-lsp-disc. A sua volta, ogni Servizio è rappresentato da una struttura dati composta, contenente tutte le informazioni reperite a riguardo: nome completo (*Istanza.Servizio.Dominio*) e le relative componenti del nome separate(utilizzate in fase di analisi dei vari dispositivi), il nodo 'target' al quale riferirsi per ottenere il servizio(e la relativa porta), e

un'elenco di campi *TXT* i quali riportano ulteriori dettagli relativi al servizio offerto, nonché del dispositivo che lo rende disponibile. Per quanto riguarda il campo contenente informazioni Dropbox del dispositivo, viene rappresentata sempre come una struttura dati composta, che racchiude gli stessi campi dei pacchetti db-lsp-disc, e in particolare la lista dei Namespaces che quel nodo della rete ha annunciato nel corso della cattura.

In primo luogo, viene effettuata un'analisi e raccolta di dati rilevanti all'interno dei pacchetti mDNS, a seconda della quale viene aggiunto o meno un nuovo dispositivo nella struttura principale, e in seguito vengono raccolte le informazioni reperite dai pacchetti Dropbox, arricchendo le informazioni sui dispositivi già rilevati con lo studio mDNS precedente, oppure aggiunti nuovi dispositivi alla struttura.

L'analisi di un pacchetto mDNS avviene a partire dal campo MAC-Address sorgente, che identifica il nodo che ha inviato il pacchetto, se presente il Layer mDNS, e in particolare la presenza di record di risposta[4] alle query: qualora uno di questi campi venisse meno, l'analisi del pacchetto mDNS si interrompe. Utilizzando quindi il MAC-Address del dispositivo, se ne verifica la presenza nella struttura principale: se già presente, viene riferita l'istanza precedentemente rilevata, altrimenti ne viene creata una nuova(ma NON ancora aggiunta alla struttura principale). Dopo aver aggiornato l'eventuale indirizzo IPv4 o IPv6 del dispositivo(che magari potrebbe esser stato riassegnato), parte il vero e proprio studio del pacchetto mDNS. Per ogni record 'risposta', viene verificato se appartiene ad uno dei tipi di record utili al nostro lavoro:

16 Record TXT: viene creata una nuova istanza del servizio, identificata dal suo nome completo (Istanza.Tipo.Dominio) e ne vengono estrapolati tutti i campi TXT contenuti nel record.

1 Record A: analogamente ai record DNS, riporta la risoluzione di un nome in un indirizzo IPv4; tipicamente, il nodo della rete che lo annuncia è esso stesso il possessore di quell'indirizzo, assegnandogli quindi il nome riportato nel record; ma può anche accadere che non lo sia: in questo caso, si scansiona l'intera struttura dati principale alla ricerca di un device che ne riporti l'indirizzo IPv4, e in caso positivo, si assegna il nome a QUEL dispositivo, non a chi lo ha annunciato.

[4]per questa analisi, è stato deciso di prendere come certa la presenza di un servizio solo se un dispositivo annuncia la presenza di tale servizio in un record 'answer'

28 Record AAAA: segue lo stesso ragionamento per i Record di tipo A, ma riportando la risoluzione di un nome in un indirizzo IPv6

33 Record SRV: vengono raccolte le informazioni contenute, quindi nome completo, nome suddiviso nei vari campi istanza tipo dominio del servizio, e l'host target al quale riferirsi.

Possono essere presenti altri tipi di record, ma ai fini di questo lavoro di tesi, non contengono informazioni rilevanti, quindi vengono ignorati.
Può accedere però, che il dispositivo che annuncia il servizio non sia effettivamente il provider di tale, ma semplicemente condivida questa informazione acquisita in precedenza. Quindi, prima di aggiungere il nuovo dispositivo con tutti i servizi rilevati, si controlla che ogni servizio attribuito al nodo abbia come target il dispositivo che lo ha annunciato: se così non fosse, viene effettuata una ricerca nella struttura principale, verificando se esiste un nodo che effettivamente è il fornitore di tale servizio(host *target*, o istanza del servizio); se non esistesse alcun dispositivo corrispondente, il servizio viene aggiunto ad una lista dove vengono raccolti tutte le istanze dei servizi i quali non hanno ancora trovato l'effettivo target al quale riferirsi, la quale a sua volta verrà controllata ogni volta che viene identificato un nuovo host, assegnando quindi i servizi al proprio fornitore.
In conclusione del recupero dati dal pacchetto mDNS, viene verificato che il dispositivo contenga informazioni rilevanti nella sua struttura: ovvero gli sia stato attribuito almeno un nome/alias che lo identifichi, o per lo meno sia l'effettivo fornitore di almeno un servizio. In caso positivo, il nodo viene aggiunto alla struttura principale.

Per quanto riguarda la raccolta di informazioni di un pacchetto Dropbox, il procedimento è semplificato, e si limita a raccogliere dati dai campi del relativo pacchetto, ed assegnare tali informazioni al relativo dispositivo nella rete locale.
Come per mDNS, per ogni pacchetto db-lsp-disc, se ne recupera l'indirizzo MAC e quello IP, se ne verifica la presenza nella struttura principale, e nel caso in cui non sia già stato rilevato, viene creata una nuova istanza di dispositivo. Proseguendo, si recuperano tutti i campi del Layer db-lsp-disc: host-int, version, displayname, port, namespaces, creando un nuovo campo del dispositivo per raccoglierle se non presente, o aggiornando la lista dei namespaces altrimenti.

Conclusa la fase di Raccolta Dati, inizia la fase di elaborazione delle informazioni raccolte, parte centrale di questo lavoro. Come per la precedente fase, vengono effettuate 2 analisi separate: una per le informazioni raccolte dal protocollo mDNS, e l'altra per lo studio delle informazioni Dropbox.

Grazie allo studio dei record mDNS, per quasi tutti i nodi della rete che hanno annunciato dei servizi, si riesce a ricavare con buona precisione la natura del dispositivo che ne ha dato disponibilità. Per etichettare un'host, è stato deciso di suddividerli in Macro-Categorie, le quali coprono in pratica la totalità dei tipi di dispositivi attualmente in circolazione:

Workstation Il gruppo più generico dei dispositivi. Qui vengono raccolti tutti i nodi che spaziano fra Personal Computer Desktop, Laptop, Server, ... e affini.

NAS É una specializzazione del gruppo Workstation, e indica tutte le macchine adibite alla funzione di Network Area Storage; sono quindi tutti quei dispositivi che archiviano e condividono dati, da file, immagini, musica, e altro.

Mobile In questa sezione vengono raccolti tutti i dispositivi mobili, quindi principalmente Smatphones, Tablets, che essi siano Apple o Android.

Printer Questo gruppo racchiude tutti quei nodi della rete che si possono categorizzare come Stampanti, Scanner, o Stampanti-Multifunzione, quindi tutti quei dispositivi usati per le utilità d'ufficio.

Media Questo gruppo indica tutti quei dispositivi utilizzati per la riproduzione di Media quali: la musica, video, foto, e qualsiasi cosa inerente all'intrattenimento in formato digitale.

Per suddividere i nodi nelle suddette categorie, sono stati utilizzati alcuni Tipi Specifici di servizi annunciati nei record mDNS, supponendo con una buona dose di certezza che dati tipi potessero essere annunciati, con buona probabilità, solo da determinati tipi di dispositivi.
Lo studio per attribuire un'identità ad un dispositivo parte con la l'analisi delle tipologie di servizi annunciati dallo stesso. In primo luogo si va alla ricerca di un particolare servizio fittizio annunciato dai dispositivi Apple chiamato '_device-info', il quale non annuncia un servizio vero e proprio, ma riporta i campi TXT *model* e opzionalmente *osxversion*: se presenti, e a meno di falsificazione e diffusione di informazioni errate da parte dell'host, si identifica con certezza e precisione la natura del dispositivo grazie a questi campi, e con l'aiuto di un dizionario riportante quasi la totalità dei numeri di modello dei rispettivi prodotti e versioni del sistema operativo Apple, si viene a conoscenza delle specifiche complete Hardware e Software della macchina. Questa, a mio avviso, si rivela l'informazione più delicata fa tutte quelle raccolte, in quanto fornisce delle specifiche veramente dettagliate e private del

nodo, fornendo ad intrusi/ospiti nella rete eventuali punti deboli per compromettere il corretto funzionamento o più semplicemente informazioni sulle scelte Hardware e Software dell'azienda in cui si trova.
Lo studio procede scorrendo l'elenco di tutti gli alias e nomi mnemonici che sono stati attribuiti al nodi stesso(nome.local) o all'istanza del determinato servizio che offre(la sezione *Istanza* del nome del servizio). Per prima cosa si va alla ricerca degli alias del tipo *nome.local*, che solitamente riporta una stringa composta, creata dalla combinazione del nome che l'utente dà alla macchina, e l'aggiunta di stringhe mnemoniche assegnate dal Sistema Operativo usato dall'utente. Questo spesso porta a creare nomi che riconducono facilmente alla natura del dispositivo, riportando keyword del tipo: 'MacBook-Pro' o 'Computer', oppure 'iPhone' e 'Android', o ancora 'Time Capsule'. Queste informazioni, se pur con un'alta dose di incertezza, sono un buon punto di partenza per supporre il tipo di dispositivo che è stato rilevato.
Ovviamente, fra tutte le informazioni raccolte e come appena accennato, questi sono dati da considerarsi non del tutto affidabili, dato che per un utente medio e consapevole, sarebbe fin troppo semplice modificare o attribuire un nome che depisti la corretta identificazione del dispositivo. Tuttavia, nell'analisi di questi nomi, si è notato che l'utente medio tende ad ignorare questo dettaglio, anzi, spesso come nome che identifichi il dispositivo utilizza il suo vero nome, talvolta seguito anche da cognome, fornendo anche questa personalissima informazione riguardo al possessore del dispositivo. Recuperare questo dato è reso possibile grazie a tutti gli utenti che come nome del dispositivo utilizzano una stringa del tipo 'Marios-iPhone.local'(in Inglese), oppure 'MacBook-Pro-di-Maria-Rossi.local'(in Italiano), dando quindi la possibilità di ipotizzare il Nome e Cognome del possessore del dispositivo.
In fine, avviene l'analisi vera e propria sui servizi annunciati dal nodo, suddivisi per categoria, e ad ognuno dei quali è attribuito un *grado di affidabilità* che spazia dal 1(completamente inaffidabile) al 9(completamente affidabile), con il quale si cerca di collocare il dispositivo in una delle Macro-Categorie sopra citate. Per ognuno dei servizi annunciati, si verifica quale categoria di dispositivi lo può aver annunciato, e con quale grado di affidabilità. Al termine di tale processo, si verifica qual'è la Categoria la quale ha la maggiore affidabilità, e se non è stato rilevato il record _device-info sopra descritto, il dispositivo, dopo un'ultimo controllo, viene categorizzato.
Prima di attribuire il nodo ad una specifica categoria, viene effettuato un'ultimo controllo, dovuto ad un comportamento anomalo riguardo l'annuncio di alcuni servizi, rilevato in fase di validazione. Alcuni nodi della rete annunciavano servizi riportando nella sezione del nome *Istanza* il carattere '@', nello specifico, la stringa era della forma: 'nome-disp-1 @ nome-disp-2'. Dopo una rapida verifica, si è raggiunta la conclusione che ad offrire il servizio

annunciato non era chi lo annunciava, ma bensì un'altro dispositivo collegato(per la totalità stampanti) alla macchina che lo annuncia e che magari non possiede l'accesso alla rete per mancanza di permessi o, più probabile, essendo sprovvisto di scheda di rete. Per identificare queste situazioni, prima di dare la conferma, si ricerca nel nome il carattere @, e si identifica il dispositivo, per esempio, non come una stampate, ma bensì come un dispositivo che condivide una stampante, aumentando quindi il grado di precisione del programma e riportando la corretta informazione.

Conclusa l'elaborazione delle informazioni ottenute dai pacchetti mDNS, si prosegue con l'analisi delle interazioni fra i dispositivi connessi alla rete locale. Riprendendo i dati contenuti nei pacchetti Dropbox, per ogni dispositivo che abbia trasmesso messaggi db-lsp-disc, si controlla se condivide nel proprio elenco di namespaces delle cartelle con altri nodi della rete locale. In caso positivo, si crea una lista associata al dispositivo che riporta tutti i nodi con cui condivide una o più cartelle Dropbox, creando così un grafo che rappresenta le interazioni fra i dispositivi e quindi fra gli utenti che usufruiscono della rete locale.
Questo a sua volta rivela anche la rete sociale degli utenti, supponendo che se persone condividono delle cartelle, è molto probabile, se non certo, che si conoscano, o per lo meno abbiano una qualche sorta di interessi o fini in comune, che siano per lavoro o per hobby.

2.3 Pro e Contro Soluzione

Analizzando nel complesso questo lavoro di tesi, si possono notare dei vantaggi e degli svantaggi nelle scelte progettuali ed implementative che adesso andremo ad analizzare.

Il primo argomento di discussione da affrontare è il punto in cui ci poniamo per raccogliere i dati da analizzare per lo studio della rete: la scheda di rete di un singolo nodo collegato alla stessa. Ovviamente questo ci pone in grande svantaggio e delle grosse limitazioni, e in particolare, ci nega l'accesso a tutti quei pacchetti unicast che non verranno mai inoltrati al nodo designato all'ascolto dei pacchetti. Per permettere l'intercettazione completa del traffico, ci si dovrebbe porre, come punto di raccolta del traffico, in uno snodo centrale della rete, come per esempio un accesspoint, uno switch o un router interno. Indubbiamente ci darebbe la visione completa del traffico, ma questo comporterebbe non poche difficoltà logistiche nell'avere accesso a

tali punti di snodo. D'altra parte quindi, questa scelta progettuale comporta una maggiore facilità nel recupero dei pacchetti che transitano nella rete locale, richiedendo quindi il semplice accesso a quest'ultima. Infatti uno degli scopi principali di questo lavoro è dimostrare la semplicità con cui reperire informazioni private dei dispositivi che vi risiedono, e come queste siano accessibili a chiunque, con il minimo sforzo: necessitando semplicemente di leggere le informazioni che ci vengono recapitate, non andandole a cercare o compromettendo la sicurezza e quindi superando i meccanismi di protezione di un dispositivo.

Un'altro punto sul quale discutere è l'utilizzo e l'affidarsi del nome mnemonico di una macchina, che se ne fa in questo lavoro per supporre la natura del dispositivo, e più in generale, l'affidabilità delle informazioni reperite all'interno dei pacchetti. Nel mondo dell'informatica qualsiasi cosa è modificabile e 'ricreabile', tanto più i pacchetti e le informazioni reperibili all'interno della rete. Basti pensare alla possibilità di formare pacchetti manualmente, seguendo gli standard dei protocolli che si vogliono imitare: si ottengono delle informazioni prodotte non dal software che implementa il protocollo, ma pacchetti che seguono il medesimo standard di layout per i campi, e che ne contengono informazioni non generate per la loro effettiva veridicità.
Tanto più è probabile che un utente accorto e consapevole di quanto siano vulnerabili le informazioni che attraversano le reti, e più in generale, i dispositivi ve vi si collegano, potrebbe decidere di associare un nome al proprio dispositivo che depisti un'analisi superficiale che tenti di scoprirne il tipo e le componenti. Basti pensare, per esempio, ad un utente che chiama il proprio Smartphone Apple iPhone con il nome 'Android', o viceversa.
Tuttavia, al termine del nostro studio e validazione dei risultati, si è riscontrata l'abitudine opposta da parte dell'utente medio: ovvero quella di non modificare affatto i nomi dei dispositivi, anzi, cercando di attribuirgli nomi ancor più evocativi e dettagliati riguardo la tipologia di dispositivo, rendendoli quindi, sì più riconoscibili dall'utente umano, ma anche facilmente identificabili da un applicativo software che ne estrapola le keyword e lo identifica. Gli esempi più rilevanti sono, oltre a quasi la totalità dei i dispositivi Apple che siano mobile o PC, sono stati nomi del tipo: nomedinodoNAS, o ancora nomedinodoPC e nomedinodo$DESKTOP$, e così via.
Come validazione di questo fatto, oltre che la verifica effettiva sulla tipologia del dispositivo, ci vengono in aiuto tutti i servizi annunciati tramite mDNS, e che quindi ne aumentano la precisione nell'identificare correttamente la categoria da attribuire al dispositivo.

Strettamente collegato al punto precedente, un'altro argomento di questo

lavoro sul quale discutere è l'affidabilità e la precisione con la quale si attribuisce una categoria ad un dispositivo, e se ne identifica quindi la tipologia. Al netto dell'incertezza dovuta alla possibilità di modificare i pacchetti distribuiti in rete, accennata in precedenza, si suppone che se un dispositivo annuncia un dato servizio, effettivamente è in grado di offrirlo. Per quanto riguarda la categorizzazione, la scelta concettuale è stata la seguente: attribuire i servizi più evocativi e discriminanti possibili ad ogni categoria di dispositivi. Ad esempio, se un dispositivo annuncia direttamente il servizio di stampa, scanner, o altro inerente a tale branca di servizi, è ovvio che il dispositivo non può essere uno Smartphone, un Laptop/Notebook o ancora un server: non ne ha fisicamente gli strumenti per offrirli! O ancora, se un dispositivo offre un servizio utilizzato unicamente da un'applicazione per Mobile, ovviamente non potrà essere un Server o una stampante. Ulteriori discussioni sono rimandate nelle pagine successive, nella sezione in cui si discute la validazione del lavoro software prodotto.

Capitolo 3

Stato dell'Arte

Con l'avvento di tecnologie come Zeroconf, alla ricerca quindi di rendere indipendenti ed automatizzate le interazioni e collaborazione fra i dispositivi e applicazioni che vi sono in esecuzione, il traffico Broadcast e Multicast è diventato sempre più diffuso, e con lui anche gli studi che sono stati approntati per migliorarne l'efficienza, l'affidabilità, la fruibilità, nonché analisi atte a comprenderne maggiormente la natura delle meccaniche che vi troviamo alla base.
Nell'elaborazione di questo lavoro di Tesi mi sono ritrovato a visionare numerosi studi a riguardo, che similmente, o con approccio completamente diverso, hanno analizzato sotto vari aspetti il traffico Broadcast e Multicast diffuso nelle reti locali.

3.1 Discovering Zeroconf Services Beyond Local Link[5]

Questo altro lavoro parte da alcune considerazioni sulla tecnologia *Zeroconf*, in particolare alla limitazione sull'utilizzo di quest'ultima all'interno della rete locale per mezzo di comunicazione multicast. É stato sviluppato un applicativo chiamato *Zeroconf-to-Zeroconf Toolkit* (z2z), nel tentetivo di realizzare a un'architettura ibrida che combinasse la semplicità di utilizzo di Zeroconf con la scalabilità di una rete peer-to-peer basata su DHT.
Z2z connette più sottoreti Zeroconf usando OpenDHT, estendendo la raggiungibilità delle applicazioni abilitate all'utilizzo di Zeroconf al di là dei link locali. Inoltre, fornisce un framework sul quale basarsi per costruire una soluzione globale di service discovery basata su Zeroconf.
Considerando l'enorme e crescente utilizzo di Zeroconf, si può notare che sempre più applicativi ne fruiscono, aumentando di molto il numero, ma so-

prattutto la tipologia di servizi offerti tramite questa tecnologia. Questo ha portato ad una considerazione: se l'utilizzo fosse limitato al processo di discovery di stampanti e dispositivi locali, non ci sarebbe alcun problema, e rivelandosi un meccanismo pienamete utilizzato. Ma ad oggi ci sono servizi come applicazioni di chat(per esempio *iChat* di Apple), i quali sono limitati da questa restrizione della rete locale: sarebbe molto più consono permettere alla comunicazione fra gli utilizzatori di Zeroconf di interagire anche con altre reti locali, mantenendo comunque tutti i vantaggi riguardo la semplicità di utilizzo.
Questo lavoro di estensione di Zeroconf è ispirato dalle recenti innovazioni nel campo di ricerca riguardo le reti peer-to-peer, tra cui le reti strutturate-sovrapposte p2p basate sulle *Distributed Hash Tables*(DHT), divenute popolari come sotto-strati sistemi distribuiti a livello globale sono basati. Le reti DHT sono caratterizzate by da un efficiente algoritmo che mappa stringhe arbitrarie in uno specifico nodo di una rete e di produrre path di routing efficienti composti da un numero limitato di passaggi fra un nodo ed un'altro. Il mapping è deterministico e uniformemente distribuito, e questo permette efficienti implementazioni di numerosi servizi di scala globale come il file-sharing e multicast-overlay.
Queste considerazioni sono state finalizzate in questo lavoro con il connettere sotto-reti Zeroconf utilizzando reti DHT, in particolare OpenDHT, un servizio DHT pubblicamente accessibile. Quindi, un processo *z2z* in esecuzione in una sotto-rete esporta i servizi Zeroconf localmente disponibili in OpenDHT. Un'altra istanza di un processo *z2z* in esecuzione in un'altre sotto-rete controlla i servizi esportati in OpenDHT e li importa nella propria rete locale, diffondendoli come se fossero originari di quella LAN. Tali servizi, agli occhi delle applicazioni, sono indistinguibili da quelli veramente annunciati localmente, e sono quindi assimilati dalle applicazioni che utilizzano Zeroconf come nulla fosse, non modificando quindi il proprio comportamento.

Questo lavoro mette ancor più in risalto quanto le informazioni diffuse tramite questi protocolli siano veramente delicate, in quanto estendendo l'area di diffusioni a più sotto-reti esterne si aumenta il numero di potenziali 'ascoltatori malintenzionati' che potrebbero usare i dati privati così ottenuti per i propri fini.

3.2 Privacy Implications of mDNS[1]

Similmente al nostro studio, questo lavoro tratta appunto le informazioni contenute nei messaggi mDNS e DNS-SD che circolano nelle reti locali, ap-

profondendo in particolare le implicazioni che hanno rispetto alla privacy degli utilizzatori di applicazioni che ne fruiscono i vantaggi.
La differenza dal nostro elaborato sta anche nella parte implementativa per quanto riguarda lo strumento per la raccolta e l'analisi delle informazioni ottenute dai record mDNS. Quest'altro strumento è stato realizzato con 3 tools aventi compiti distinti: il primo, chiamato *Logger*, ha il compito di richiedere e scoprire la presenza di servizi annunciati tramite mDNS nella rete locale, stimolando la comunicazione dei dispositivi connessi effettuando delle querys attive per verificarne la presenza; un *Database Server* il quale raccoglie le informazioni ottenute a seguito delle querys, archiviandole permanentemente in un Database; un *Analyzer*, il quale recupera le informazioni raccolte nel Server Database e le analizza, filtrandole per tipo di servizio/protocollo, e producendo l'effettivo studio dei dati raccolti.
Le comunicazioni fra il Server e gli altri 2 componenti sono state implementati tramite comunicazioni REST, quindi il server offre una semplice interfaccia supportata dal protocollo HTTP con la quale interagire. A tale scopo, è stato utilizzato il framework Spring-Boot[10], che è un'implementazione REST.

Per quanto riguarda l'analisi, questo altro lavoro si è concentrato non tanto sulla scoperta di informazioni del singolo dispositivo, ma bensì sull'analisi e lo studio dei singoli servizi/protocolli annunciati all'interno della rete, disinteressandosi dallo scoprire le caratteristiche del singolo nodo. Un'altra differenza nell'analisi sta nel fatto che è stato effettuato uno studio separato fra varie tipologie di rete(Privata, Semi-Pubblica, Pubblica/Open), confrontando poi le varie tipologie di protocolli rilevate, e in quali percentuali nelle une rispetto che nelle altre. Il nostro lavoro invece si è specializzato nel rivelare identità del dispositivo e, in alcuni casi, del possessore, fruendo delle sole informazioni reperibili in modo completamente passivo, e quindi non generando alcun tipo di traffico per stimolare l'invio di messaggi mDNS.

3.3 How Broadcast Data Reveals Your Identity and Social Graph[8]

Un'altro studio simile a questo lavoro di tesi è l'analisi del traffico Broadcat e Multicast effettuato dall'Università di Scienze Applicate di Asburgo, la quale ha ampliato lo studio arricchendolo con l'aggiunta di altri protocolli oltre a mDNS e Dropbox, il tutto incrociando i risultati con i Databases interni dell'università, identificando con precisione gli utenti, rivelandone completamente l'identità.

Inoltre, il team di Asburgo si è basato su di una quantità di dati molto più elevata rispetto al nostro studio, se pur limitandosi alla rete interna all'università: ben 40GB di traffico broadcast per ogni 6 mesi, e per un totale di 2 semestri di monitoraggio.
Oltre ai protocolli usati in questo lavoro, sono stati analizzati e sfruttati per la raccolta di ulteriori informazioni, i pacchetti generati da protocolli come SSDP, LLMNR, e NetBIOS. Questo gli ha reso possibile accrescere il numero di informazioni ottenute, con dettagli che spaziano fra ulteriori nomi degli host grazie a NetBIOS, alla lingua parlata dall'utente(Inglese, Tedesco, ...) grazie sempre al parsing del nome dei nodi del tipo 'iPhone von John Doe', il quale indica chiaramente le origini Tedesche dell'utente grazie alla keyword 'von'.
In fine, questo altro studio ha incrociato gli hostnames raccolti con i dati nel server LDAP dell'università, il quale contiene l'intero elenco di tutti i nominativi(nome e cognome), matricole, email, corsi tenuti/seguiti, status(studente, insegnate, personale...) di tutti gli utenti registrati e facenti parte dell'università. Per disambiguare alcuni 'omonimi' fra i vari nomi estratti dagli hostnames rilevati e i nomi degli utenti registrati, sono stati utilizzati anche il timestamp del momento della cattura delle informazioni, incrociandoli con i corsi seguiti dagli utenti, tenuti dai professori, o ancora con i turni di lavoro del personale, associando al dispositivo, nella maggior parte dei casi, il corretto proprietario.

3.4 Arginare il problema della Privacy

Presa coscienza dei problemi che comporta la trasmissione in chiaro delle informazioni diffuse tramite Broadcast/Multicast, e quindi mDNS in particolare, sono stati approntati numerosi studi atti ad aggiungere un layer di sicurezza e information-hiding, rendendo tali dati fruibili soltanto agli utenti autorizzati.
Fra i vari lavori a riguardo, mi sono trovato ad osservarne alcuni, che di seguito saranno brevemente accennati.

Daniel Kaiser and Marcel Waldvogel, autori dell'elaborato *Adding Privacy to Multicast DNS Service Discovery*[6], hanno ideato un'estensione nelle comunicazioni DNS-SD per la privacy, la quale permette di nascondere tutte le informazioni sensibili dei record che annunciano i servizi offerti da un host, non necessitando alcuna modifica nella configurazione della rete, limitandosi all'aggiunta di una piccola fase di *pairing* iniziale.
Dopo la solita analisi di quali informazioni si possono ricavare dai messag-

gi DNS-SD, è stato deciso di sviluppare uno strumento che, tramite tecniche di crittografia, non appesantisse il protocollo con l'aggiunta di overhead computazionale, e inoltre non comprometta il regolare funzionamento delle applicazioni che usufruiscono del protocollo di Service Discovery, ma che riuscisse comunque a fornire l'oscuramento delle informazioni per tutti gli host non autorizzati. Per permettere ciò, è comunque necessaria una piccola fase iniziale di *pairing* fra i dispositivi che vogliono condividere/richiedere il permesso per l'utilizzo del Servizio: questo è stato implementato tramite lo scambio di informazioni per mezzo di un canale sicuro/esterno alla rete locale, come ad esempio Bluetooth, un QR-Code o NFC, o ancora una email crittografata o un SMS. In questa fase, i 2 host si scambiano tutte quelle informazioni necessarie a decrittare i dati diffusi in rete, rendendo personalizzabile, da parte dell'host che offre il servizio, la decisione di condividere o meno informazioni con gli altri nodi della rete.

In un'altro lavoro, *User-Friendly, Versatile, and Efficient Multi-Link DNS Service Discovery*[2] è stato elaborato un modo per fornire capacità multi-link e scalabilità per DNS-SD pur mantenendo la sua semplicità d'uso e l'efficienza. Viene quindi proposta una soluzione Stateless-DNS (DNS-SD/sDNS), che permetta l'autoconfigurazione di Service Discovery spazi di nomi arbitrari, largamente indipendenti dal layout fisico della rete, sfruttando la loro tecnica *statles DNS* e l'algoritmo di consensus Raft
Non mi sono addentrato nei dettagli di quest'ultimo studio, quindi per ulteriori informazioni riferirsi alla bibliografia di questo elaborato.

Capitolo 4

Dettagli di Implementazione

Questo lavoro di tesi, nello specifico lo sviluppo del componente software, è stato reso possibile, oltre a Wireshark, grazie ad un wrapper di tshark scritto in python chiamato *pyshark*, il quale fornisce delle funzioni d'interfaccia che permettono di catturare/leggere file di cattura, ed accedere ai campi che compongono i vari pacchetti, estraendo informazioni utili.
Lo strumento software invece è stato sviluppato in Python nella versione 3, ed è composto da 4 moduli: DropBox_utils.py, classFrames.py, discriminators_sets.py, run_test.py, i quali verranno descritti in dettaglio in questo capitolo.

4.0.1 *WireShark* (*TShark*)

Software Open-Source che permette di catturare il traffico di dati che transita sulla scheda di rete della macchina sulla quale viene mandato in esecuzione, senza la necessità porsi in punti di snodo "centrali", come Router o Accesspoint. Questo limita molto il traffico dati catturabile, ma permette comunque di ottenere informazioni su tutti i pacchetti inviati/ricevuti a altri dispositivi in Unicast della macchina sulla quale è in esecuzione Wireshark, ed quindi anche tutti i messaggi inviati/ricevuti in Broadcast/Multicast, che per i nostri scopi è più che sufficiente. Wireshark riesce a "comprendere" la struttura di diversi protocolli di rete, è in grado di individuare eventuali incapsulamenti, riconosce i singoli campi e permette di interpretarne il significato. Per la cattura dei pacchetti Wireshark non dispone di proprio codice, ma utilizza libpcap/WinPcap, quindi può funzionare solo su reti supportate da libpcap o WinPcap.
??? Mette in condizioni di non rendere visibile al resto della rete che si sta analizzando del traffico dati ... ???.

Tshark, nello specifico, è una utility "a linea di comando" di Wireshark, che utilizza quindi il core del programma principale, offrendo le medesime funzionalità, ed è principalmente usato in tutti quei casi in cui non si necessita di interfaccia grafica, e si richiede output standard(in-line)

4.0.2 *Pyshark*

Wrapper per tshark, reperibile sulla piattaforma GitHub ... ??? . Non è propriamente un dissector, come molti altri, ma si limita a usare la funzionalità di tshark di esportare XMLs per usare il loro parsing. Questo package permette il parsing sia da file di cattura, sia da "catture Live", utilizzando tutti i dissector di wireshark installati sulla stessa macchina.
Ai fini di questo lavoro, è stato usato il parsing su file di cattura precedentemente ottenuti, in quanto un analisi della rete il più completa possibile si ha soltanto dopo un certo quantitativo di pacchetti catturati.
Ogni pacchetto viene rappresentato da una struttura dati chiamata *Packet*, la quale suddivide il pacchetto in *Layer*, accessibili tramite indice o "nome". Un *Layer* contiene tutti i campi che un "layer di un pacchetto" possiede, sotto forma di *LayerField*. In fine, un *LayerField* contiene, nel caso in cui il campo contenga una singola informazione, tutti i dati contenuti in un campo di un layer di un pacchetto; nel caso in cui il campo contenga un gruppo di informazioni, per lo più strutturate a sua volta, tale campo viene suddiviso in più *LayerField*s, ognuno dei quali contiene una lista ... un gran casino ??????????????????????????????

4.0.3 Python 3

Per lo sviluppo del tool di analisi, è stato scelto come linguaggio Python nella versione 3,un linguaggio multi-paradigma, che ha tra i principali obiettivi dinamicità, semplicità e flessibilità. Uno dei motivi che ha portato a sceglierlo come linguaggio da usare, oltre al fatto che supporta la programmazione Object Oriented che permette una maggiore modularità in fase di progettazione e implementazione, è stato il suo supporto nativo a strutture ...??? ... hash-table, qui chiamate *dizionari*. Il Dizionario è la struttura che sta alla base dell'intero tool, e permette di raccogliere informazioni ed accedervi rapidamente e intuitivamente per mezzo delle proprie "chiavi".
Inoltre, Python supporta anche un modo d'uso interattivo attraverso il quale è possibile inserire codice direttamente da un terminale, visualizzando immediatamente il risultato. Questo strumento è stato fondamentale, oltre che in fase di sviluppo e di debug, anche a comprendere il funzionamento e le

strutture con le quali *Pyshark* raccoglie le informazioni estratte all'interno dei pacchetti dati in input tramite i file di cattura.

4.1 Il Tool di Analisi: Struttura

Lo strumento sviluppato per la raccolta e l'analisi delle informazioni rilevanti sui dispositivi connessi alla rete è stato strutturato in 3 moduli, ognuno dei quali implementa determinate funzionalità:

classFrames.py nel quale sono implementate tutte le classi utili a raccogliere e contenere le informazioni in modo strutturato; inoltre alcune classi implementano le funzioni che svolgono il lavoro di analisi: caratterizzazione e catalogazione dei dispositivi, e interazioni con gli altri nodi della rete.

DropBox_util.py qui viene implementata, nello specifico, la struttura dati che contiene le informazioni ricavate dai pacchetti Dropbox, e quindi tutti i campi dei messaggi db-lsp-disc.

disc...tors_sets.py in questo modulo sono raccolte tutte quelle strutture(*insiemi* e *dizionari*) utilizzate per categorizzare i dispositivi nelle varie tipologie scelte in questo lavoro. Inoltre, sono presenti anche dizionari utilizzati per identificare con precisione i dispositivi tramite codice di modello e Sistema Operativo, e dizionari di "keyword" spesso ricorrenti all'interno degli hostnames rilevati, anch'essi usati per attribuite un tipo e un ipotetico proprietario al dispositivo.

run_test.py Questo è il modulo mandato in esecuzione per l'analisi completa del traffico di rete, ed è costituito da una sequenza di istruzioni che, in breve, apre i file di cattura utilizzati per testare l'algoritmo, li analizza passandoli alle funzioni definite negli altri moduli, ed infine stampa a linea di comando i risultati, componendo un'elenco di dispositivi "categorizzati", seguito da una lista che rappresenta i collegamenti che i suddetti dispositivi hanno fra di loro.

Adesso andremo ad analizzare in dettaglio ogni singolo modulo del tool.

4.1.1 classFrames.py

In questo modulo, come già accennato, sono presenti tutte le classi utilizzate per raccogliere in maniera strutturata le informazioni mDNS e analizzarle insieme ai dati ricavati dal protocollo Dropbox

La classe ***Target*** è rappresentazione strutturata di un target, con le relative porte, al quale riferirsi per fruire di un servizio annunciato nella rete locale, quindi: *nomeHost.local* e *#Porta* per connettersi al servizio. Offre metodi per aggiornare il numero delle porte e getter per reperire le informazioni.

La classe ***ServiceMDNS*** racchiude tutte le informazioni carpite dai vari record DNS sparsi in tutti i pacchetti reperiti nella rete, quindi che vanno dal nome completo che identifica Istanza e Tipo, ai campi *TXT* opzionali che forniscono dettagli su di esso. Oltre alle funzioni di *add_** e *update_** per aggiornare i dettagli del servizio offerto, è stata implementata una funzione interna di utilità, usata per scomporre il nome "composto" del servizio, ottenendo i campi separati di *Istanza*, *Tipo Servizio*, e *Dominio*(e Protocollo usato). Quest'ultima funzione di utilità è mutuata dal fatto che spesso il dissector usato non separa correttamente queste sezioni del nome, solitamente a causa della stringa della sezione *Istanza* che contiene il carattere ".", inducendo ad una scorretta scomposizione. Per arginare questo problema, nella funzione implementata, la stringa con il nome completo viene "dissezionata" a partire dalla fine di essa, confidando nel fatto che non è presente il carattere "." all'interno delle sezioni *Dominio* e *Tipo Protocollo*

La classe ***Device*** è la rappresentazione strutturata di tutte le informazioni ricavate su di un dispositivo. I campi che lo compongono sono: ***id***, che ne indica il suo MAC-Address; ***kind***, che ne rivela la tipologia, e lo si ottiene a seguito dell'analisi di tutti i servizi offerti da questo particolare nodo; ***owner*** ne indica il presunto possessore del dispositivo, ricavato dall'elaborazione dell'hostname.local; ***lastIPv4*** e ***lastIPv6*** sono gli ultimi indirizzi IP v4/v6 che sono stati associati al nodo(al MAC-Address); ***services*** è una hash-table nella quali sono raccolti tutti i *ServiceMDNS* che sono stati annunciati dal dispositivo, e sono identificati univocamente dal loro nome completo *Istanza.Tipo.Dominio*; ***alias*** è un'nsieme dove vengono raccolti tutti gli alias con i quali viene identificato il dispositivo, ovvero dall'hostname.local, ai *nomi_Istanze* dei vari servizi offerti; ***db_lsp_disc*** contiene la struttura dati custom(descritta nel modulo Dropbox_utils.py) che racchiude tutte le informazioni ricavate dai pacchetti Dropbox, in particolare contiene l'elenco aggiornato di tutti i namespaces annunciati fino a quel momento.
Seguono una serie di metodi per aggiornare/aggiungere informazioni riguardanti il dispositivo. In particolare, ***add_alias*** è un metodo che, oltre che aggiungere una stringa all'insieme di alias, "ripulisce" la ripulisce e ne riporta il corretto appellativo rimuovendo la parte in cui indica "per quale altro dispositivo sta offrendo il servizio", ovvero: se un Laptop sta condividendo i servizi di una stampante ad esso collegato, l'alias ricavato dalla stringa *Istan-*

za_del_servizio sarà del tipo:
"*StampanteCondivisa @ Laptop_hostname*"
ottenendo come vero alias "*Laptop_hostname*".
Il metodo usato per attribuire un tipo al dispositivo è chiamato *update_kind*, e il suo unico scopo è quello di richiamare i metodi di un'altra casse custom, *WhoIsWhat*, descritta a breve.

La classe ***NetworkLAN*** implementa la struttura principale che contiene la lista di tutti i dispositivi riconosciuti, e le loro interazioni.
Principalmente è composta da ***devices***, un dizionario con i dispositivi aventi come chiave il loro *id*, e *dorpboxSubNET*, un dizionario(sempre con chiavi gli id dei dispositivi) che contiene le liste di tutti i nodi con cui un dispositivo condivide cartelle Dropbox. Per chiarire il concetto, il dizionario ***dorpboxSubNET*** ha la seguente struttura:

"id 1" : {Insieme di *id* di tutti i dispositivi con i quali condivide una cartella}

"id 2" : {Insieme di *id* di tutti i dispositivi con i quali condivice una cartella}

"id 3" : ...

Gli altri due dizionari che fanno parte della classe, *lost_srv_propertiese* e *namespaces_in_common*, sono delle strutture di supporto ai metodi che adesso andremo ad analizzare:

- ***extract_mDNS_info*** è il metodo che dato un pacchetto, ne ricava le informazioni mDNS utili al nostro lavoro. Il metodo implementa l'algoritmo a pagina 19, nella sezione in cui viene descritta l'*analisi di un pacchetto mDNS*. Per quanto riguarda l'implementazione, sono tate utilizzate numerose strutture di utilità(liste) a causa della complessa struttura interna con la quale è stata implementata la classe *Packet* fornita da *Pyshark*: ogni record di risposta NON viene contenuto in una singola struttura dati, ma viene diviso in vari *LayerField*s, ognuno dei quali contiene una lista con tutti i sotto-campi dello stesso tipo.
  Per chiarire:
  Se un pacchetto contiene un'array di record DNS di risposta[1], ogni record NON è rappresentato da un singolo LayerField contenente tutti i campi di un *record DNS*, ma bensì ci saranno tanti LayerField(quanti sono i campi del *record DNS*) che conterranno ognuno una lista campi del record dello stesso tipo, ottenendo quindi: lista di recod.nomi, lista

[1][Nome | Tipo | Classe(IN) | TTL | Lunghezza | Dati]

di recod.tipi, ... e così via.
Questa strutturazione ha comportato un'enorme lavoro per raccolta dati mDNS, e le scelte implementative dell'algoritmo sono state basate su di essa. Come ad esempio il motivo per cui la raccolta di informazioni che risiedono nei record SRV è separata da quella dei record TXT, A, o AAAA, in quanto il nome dei record SRV, come i suoi campi, sono situati in una lista separata dagli altri.
Per concludere la descrizione di questo metodo, si rimanda la visione al prossimo punto nel quale viene descritto il processo con il quale viene "ripulito" il dispositivo da tutti quei servizi che annuncia, ma che non offre direttamente.

- ***cleanup*** è un metodo di utilità che, dato un dispositivo, verifica che tutti i servizi da lui annunciati sono a sua volta offerti dallo stesso. Per verificarlo, se ne controlla il *target* a cui riferirsi per usufruire del servizio, che non è però sempre disponibile: per arginare il problema, viene creato un "*target fittizio*", ricavato dalla sezione *Istanza* del nome del servizio, che comunque identifica un dispositivo dato che fa anche parte dei vari alias di quest'ultimo; nel caso in cui il servizio sia *condiviso*, e quindi riporti nel nome dell'*Istanza* il carattere @, viene elaborato come già descritto nella classe *Device.add_alias()* in questa sezione.
  Proseguendo, se un servizio non è offerto direttamente dal dispositivo, prima si controlla se è stato rilevato un nodo che ne sia il target(controllando fra i suoi alias): in caso positivo si assegna correttamente il *ServiceMDNS*, altrimenti lo si inserisce nella struttura interna *lost_srv_propertiese* per mezzo del metodo *add_lost_property*.

- ***add_lost_property*** si limita ad aggiungere un servizio nella struttura interna *lost_srv_propertiese*, indicizzandolo per ogni *Target* attribuito a quel servizio.

- ***search_lost_propertyes***: metodo che, dato un dispositivo, controlla per ogni suo alias se vi siano servizi indicizzati nella struttura *lost_srv_propertiese*, e li aggiunge ai servizi da lui offerti.
  Questo metodo è usato ogni volta che viene aggiunto un nuovo alias al dispositivo, in modo tale da recuperare eventuali servizi che non sono potuti esser stati assegnati perché magari non era stato ancora associato un particolare alias.

- ***extract_DB_infos***, dato un pacchetto, ne ricava le solite informazioni di id(MAC-Addres) e IP v4/v6 del dispositivo, ma non implementa l'estrazione delle informazioni ricavabili da un pacchetto Dropbox: questo

è delegato al metodo costruttore della classe *DBlspDISC*, implementata nel modulo *Dropbox_utils.py*.

- ***get_dropbox_subNET***: metodo che ha lo scopo di verificare quali dispositivi condividono cartelle Dropbox con gli altri nodi della rete, e collegarli fra loro, riportando i risultati nella struttura *dorpboxSubNET* e quindi rappresentando la "sottorete" che si viene a creare fra gli utenti Dropbox.
  Il processo si divide in 2 fasi: nella prima si creano delle liste, raccolte nella struttura *namespaces_in_common*, indicizzate dal "codice" dei *namespace* annunciati nei pacchetti *db-lsp-disc*, all'interno delle quali si trovano tutti gli *id* dei dispositivi che li hanno diffusi; nella seconda fase, per ogni *namespace*, se ne scorre la lista di *id*, per ognuno si crea un'ulteriore lista di id, nella struttura *dorpboxSubNET* che come descritto in precedenza, elenca tutti i dispositivi con i quali quel nodo della rete condivide almeno una cartella.

I restanti metodi sono semplici funzioni di utilità per una stampa, formattata a linea di comando, utilizzati per visualizzare all'utente i risultati ottenuti dallo studio.

La classe ***WhoIsWhat*** implementa tutta la parte di *Analisi* delle informazioni raccolte dai pacchetti mDNS, e tenta di classificare il dispositivo dato in input al costruttore della stessa.
Contiene delle strutture dati di supporto, quali:
protos: contiene l'insieme di tutti i *tipi* dei servizi annunciati;
bestMatches: insieme di categorie a cui potrebbe appartenere il dispositivo sotto analisi;
kindPool: dizionario che ha come indici le categorie di dispositivi, e come valore associato un intero che indica il "grado di fiducia" con il quale il dispositivo appartiene a l'una o l'altra categoria;
kind: Risultato dell'analisi, qui viene riportata la categoria che meglio descrive il dispositivo;
rel_lev: livello di affidabilità massimo incontrato;
guess_owner: qui viene riportato il nome dell'ipotetico proprietario ricavato dal host-name;
L'Algoritmo è implementato[2] nel metodo ***check***, e i metodi all'interno utilizzati: *check_dev_info*, *check_on_local_alias* e *check_MDNS_proto*.
L'analisi parte scorrendo l'insieme di servizi annunciati dal dispositivo, durante il quale si verifica la presenza del "servizio fittizio" *_device-info*. Se

[2]Utilizzando i dizionari definiti nel modulo *discriminators_sets.py*

presente viene utilizzato il metodo ***check_dev_info***, il quale non fa altro che recuperare i codici di modello e versione OS dai relativi campi *TXT*(*model* e *osxversion* se presente), e verificare se nelle strutture interne *apple_products* e *apple_osx_versions*[3] sia presente un dispositivo indicizzato con quel/quei codice/i, identificando quindi con precisione il dispositivo.
Proseguendo nell'algoritmo, si analizzano gli alias del nodo, in particolare quelli della forma *alias_dev.local*, che ne indicano l'hostname, utilizzando il metodo ***check_on_local_alias***: la sua funzione è quella di verificare se è presente una delle *keyword* situate nel dizionario *keyword_on_alias*, e in caso positivo assegna la tipologia del dispositivo associata a quella keyword; in oltre, questo metodo tenta di dedurre il possessore del dispositivo, che spesso viene riportato insieme alla parola chiave nell'alias, come accennato nel Capitolo 2.
Viene poi approntato lo studio sui singoli servizi annunciati, con il metodo ***check_MDNS_proto***: per ogni servizio si controlla se quel servizio appartiene[4] ad una categoria di dispositivi che solitamente lo annuncia; il controllo avviene in 2 fasi: nella prima si controlla se il servizio appartenga al'insieme di quelli che sicuramente vengono annunciati esclusivamente da un particolare tipo di dispositivo, e se non ne viene rilevato nessuno, allora si passa alla seconda fase, nella quale si allarga il campo controllando in insiemi(sempre indicizzati per categoria) che riportano i servizi probabilmente annunciati da quella categoria, e "il grado di fiducia" con il quale quel servizio è annunciato da quella categoria di dispositivi. Alcuni esempi verranno presentati nella sezione in cui verrà descritto il processo di *validazione del tool*.
In fine, con il metodo ***get_kind***, prima di restituire la stringa che riassume l'analisi del dispositivo, vengono effettuati alcuni passaggi per "ripulire" e correggere le informazioni raccolte da eventuali dati che potrebbero indurre ad una errata categorizzazione del dispositivo. In particolare tramite il metodo ***shared_printer***, si controlla se il servizio che ha indotto ad attribuire l'etichetta "Stampante" a questo dispositivo sia un servizio che viene direttamente offerto da quest'ultimo: il caso opposto viene rilegato grazie alla presenza del carattere "@" presente nel *nome dell'Istanza* del servizio, come già descritto nel Capitolo 2. Al termine di questo processo, si compone una stringa che elenca, separate da "/", tutte le caratteristiche ricavate da questa analisi, che il dispositivo potrebbe avere.

[3] in quanto questo particolare "servizio" è sfruttato solo dai dispositivi Apple
[4] Tramite i dizionari *ALLprot* e *SPECprot*

4.1.2 DropBox_util.py

All'interno di questo modulo è implementata la classe ***DBlspDISC***, che rappresenta una struttura dati che contiene le informazioni recuperate da un pacchetto Dropbox *db-lsp-disc*.

Si compone di alcune variabili e strutture dati interne, usate per contenere le informazioni reperite dal pacchetto: *host_int*, *displayname*, *version* e *port* sotto forma di stringhe; e la più importante per il nostro studio, *namespaces*, implementato per mezzo di un'insieme, e riporta l'elenco dei namespaces, e quindi delle cartelle Dropbox, che l'host ha in condivisione in quel momento. Sono inoltre presenti alcune costanti usate per accedere ai campi.

L'intera funzionalità della classe è implementata nel suo costruttore ***__init__()***, il quale passatogli un pacchetto[5] recupera l'oggetto json all'interno del quale sono riportate le informazioni da recuperare. Ottenuto l'oggetto Json, ne scorre i campi, salvandosi tutte le informazioni reperite all'interno.
I restanti metodi della classe sono dei semplici *getter*, ad eccezione di ***update_ns()*** che aggiorna la lista dei *namespaces* di un dispositivo aggiungendone di nuovi.

4.1.3 discriminators_sets.py

In questo modulo sono raccolte tutte le strutture dati utilizzate per "discriminare" e categorizzare le varie tipologie di dispositivi.

La classificazione generale dei dispositivi connessi d una rete locale è stata ridotta a 5 "MACRO-Aree": in fase di progettazione è stato deciso di non specializzarsi ulteriormente in Classi più specifiche, in quanto, dato le informazioni prese in esame, si rischierebbe di incorrere ad una categorizzazione errata e non accurata come quella utilizzata per questo studio.
Le "*Classi*" con le quali si suddividono i dispositivi sono: *WORKSTATION*, *NAS*, *PRINTER*, *MOBILE*, *MEDIA*.
Per quando riguarda la caratterizzazione di ogni gruppo si rimanda al Capitolo 2, dove viene descritto ogni Categoria.

Le suddette categorie sono tate usate per indicizzare gli insiemi di protocolli che li caratterizzano in 2 strutture dati(*hash-table* o come vengono

[5]Nota: in questo caso il pacchetto che gli viene passato deve esser stato ottenuto da una funzione del modulo PyShark con l'opzione/argomento "*use_json=True*", altrimenti i dati da recuperare si trovano in un formato illeggibile

chiamati in Python *dizionari*): ***ALLprot*** che è il più generico dove sono raccolti, raggruppati per categoria, tutti quei servizi che "*molto probabilmente*" vengono annunciati da quella categoria di dispositivi, e ad ogni servizio è associato un "*grado di fiducia*" con il quale si indica la probabilità che quel servizio sia annunciato da quel tipo di dispositivo; ***SPECprot*** è l'hash-table, sempre suddivisa come la precedente, che riporta i servizi suddivisi per categoria che sicuramente(o quasi) sono annunciati esclusivamente da quella "specifica" categoria di dispositivi.
La scarsa quantità di documentazione riguardo i servizi DNS-SD ha portato ad un considerevole lavoro per capire quali vengano annunciati da alcune categorie rispetto che ad altre. Questo è dovuto anche al fatto che molti di essi non siano veri e propri *standard*, ma bensì proprietari, impedendo uno studio approfondito riguardo il loro utilizzo e le modalità con cui venissero annunciati. Anche per tutti quei servizi indicati come "standard", spesso si incorre in situazioni poco chiare riguardo al "chi annuncia quel servizio", o magari se quel determinato servizio viene annunciato "per conto di altri", come nel caso affrontato per le stampanti. Lo studio quindi ha necessitato di un pò di deduzione logica e interpretazione umana per assegnare un servizio ad una categoria rispetto che ad un altra, non limitandosi alla pura "appartenenza ad una classe", ma come descritto nella sezione precedente, è stano necessario una certa elaborazione, in alcuni casi specifica al determinato servizio. Ulteriori dettagli verranno discussi nella descrizione della fase di validazione.

Altre 2 hash-tables, ***apple_products*** e ***apple_osx_versions***, sono invece state utilizzate per identificare con esattezza modello e Sistema Operativo dei dispositivi Apple che diffondono il "servizio" *_device-info*, e che riporta nei record TXT associati: il numero del modello e opzionalmente l'identificativo della versione dell'OS che in quel momento è in esecuzione su quella macchina. Grazie a ntop??? ...

Le ultime strutture sono adibite all'analisi e alla scomposizione dell'*hostname.local* annunciati dai dispositivi nei record di risposta mDNS di tipo A o di tipo AAAA.
La prima è un'insieme di *dizionari*, sempre indicizzati dalle Categorie di dispositivi precedentemente descritte, chiamato ***keyword_on_alias***, all'interno del quale, per ogni categoria, è presente un ulteriore dizionario che contiene le coppie <*chiave, valore*>, dove *chiave* rappresenta appunto la keyword che solitamente è presente negli hostname di quel tipo, e *valore* è una stringa che descrive brevemente quale è il dispositivo che riporta quella keyword nel proprio nome.
Le ultime 2, ***common_string*** e ***common_string_s*** sono liste ordinate di

"stringhe comuni" che solitamente compongono l'hostname.local, e sono utilizzate per "ripulire" la stringa ed ottenere un *nominativo* al quale attribuire il ruolo di "possessore del dispositivo". Questo processo è stato una conseguenza dello studio dei vari hostnames annunciati nelle reti utilizzate per la cattura di traffico dati in fase di sviluppo, test, e validazione, raccogliendo quindi i pattern di stringhe più comuni ed ottenendo dei risultati tutto sommato accettabili in fase di validazione, e identificando nella maggior parte dei casi il nome corretto del possessore del dispositivo.

4.1.4 run_test.py

Modulo utilizzato principalmente in fase di *test* e di *debug*, ma che tutto sommato ne è un "*template*" dal quale se ne può ricavare un tutorial di utilizzo dello strumento sviluppato per questo lavoro di tesi.

Non è altro che una sequenza di istruzioni, le quali creano un'istanza della classe principale *NetworkLAN*, aprono una serie di file di cattura tramite la funzione offerta da Pyshark *pyshark.FileCapture()* nei 2 rispettivi modi[6] con cui codifica i pacchetti restituiti: *use_json=False* per recuperare le informazioni mDNS, e *use_json=True* per ottenere i dati nei campi dei pacchetti Dropbox db-lsp-disc.

4.2 Validazione

Il processo di validazione ...

[6] [!] come accennato in precedenza, l'utilizzo di ognuna delle 2 opzioni implica la possibilità di recuperare facilmente le informazioni di un tipo(mDNS/Dropbox), impossibilitando il recupero delle informazioni dell'altro. Per ulteriori dettagli, riferirsi alla documentazione di PyShark reperibile in rete

Capitolo 5

Conclusioni

Scrivere le conclusioni della tesi.

Bibliografia

[1] Winfried Baumann. Auswirkungen von mDNS auf die Privatsphäre. (German) [privacy implications of mdns]. *Thesis in Informatics: Games Engineering*, February 15, 2017.

[2] Holger Strittmatter Oliver Haase Daniel Kaiser, Marcel Waldvogel. User-friendly, versatile, and efficient multi-link dns service discovery. *IEEE, 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops*, pages 146–155, 27-30 June 2016.

[3] Network Working Group. Protocol standard for a netbios service on a tcp/udp transport: Concepts and methods. *Request for Comments*, 1001, March, 1987.

[4] M. Krochmal Apple Inc. Internet Engineering Task Force (IETF), S. Cheshire. Multicast dns. *Request for Comments*, 6762, February 2013.

[5] Wolfgang Kellerer** Jae Woo Lee*, Henning Schulzrinne* and Zoran Despotovic**. *z2z: Discovering Zeroconf Services Beyond Local Link.*
* Department of Computer Science, Columbia University, New York, USA
** DoCoMo Communications Laboratories Europe, Munich, Germany.

[6] Daniel Kaiser and Marcel Waldvogel. Adding privacy to multicast dns service discovery. *IEEE, 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 809–816, 24-26 Sept. 2014.

[7] Mukesh Kesharwani. Understand the computer browser service. *kesharwani.blogspot.com/*, Friday, March 18, 2011.

[8] Fabian Weisshaar Michael Faath, Rolf Winter. How broadcast data reveals your identity and social graph. *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2016.

[9] [On-line]. Service name and transport protocol port number registry.[https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml].

[10] Pivotal Software. Spring boot. [online]. available: https://projects.spring.io/spring-boot/.

Appendice A

Titolo dell'appendice

Se lo si ritiene necessario, scrivere una o più appendici.

Appendice B

Titolo dell'appendice

Se lo si ritiene necessario, scrivere una o più appendici.