Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Servizi Onion
Dalla teoria all'implementazione

Laureando
Leonardo Migliorelli

Matricola 113920

Relatore
Fausto Marcantoni

Correlatore
Correlatore

A.A. 2022/2023

Indice

Listings

Elenco delle figure

Elenco delle tabelle

1. Introduzione

Le moderne tecnologie di rete consentono una rapida comunicazione da ogni parte del mondo, avvicinando culture altrimenti distanti migliaia di chilometri. Due dei più grandi temi del nostro secolo sono la privacy e l'anonimato, in particolare parlando di reti internet ogni connessione tra client e server passa per una moltitudine di router che conoscono esattamente l'indirizzo (e quindi l'identità) del mittente e del destinatario. Con un pò di conoscenze non è complicato scoprire questi dati e sfruttarli a proprio vantaggio, le stesse big company spesso usano l'indirizzo IP con cui ci si connette al loro sito per tracciare l'utente e fornirgli articoli e pubblicità mirata o vendere i medesimi dati a terzi. La Rete Onion è stata creata per risolvere esattamente questo problema, implementando le giuste tecnologie per proteggere gli utenti dall'analisi del traffico e dalle intercettazioni

1.1 Motivazione

1.2 Obiettivi

La tesi ha come principale obiettivo la dimostrazione di come è possibile generare un servizio/web server che sfrutta la rete tor per rendere le connessioni anonime e rendere anonimo lo stesso server, creare un hostname Tor personalizzato, far si che il servizio sia raggiungibile attraverso un motore di ricerca onion e fornire un sistema di pagamento integrato per i prodotti del servizio

1.3 Struttura della Tesi

Il percorso della tesi parte dalle mix networks passando per la prima versione di Onion fino a Onion v3 e la successiva implementazione di un servizio completo

2. Onion Routing

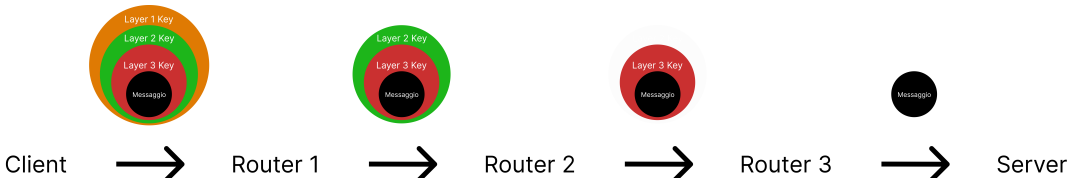

Figura 2.1: Vita di un pacchetto onion

La **rete onion** è una rete distribuita composta dall'insieme di **router onion** che agiscono come nodi di rete e collaborano per portare un pacchetto dalla sorgente alla destinazione. Il tutto avviene senza che nessun nodo possa conoscere contemporaneamente l'host sorgente e l'host di destinazione, grazie alla criptografia a strati del pacchetto, per cui ogni strato viene criptato con una chiave differente e può essere decriptato solo dal nodo con la stessa chiave simmetrica, scoprendo cosi le informazioni sul prossimo nodo. Il nodo finale (exit node) può infine decriptare l'intero messaggio e scoprirne il corpo. Questo meccanismo è derivato dallo studio di David Chaum riguardo alle mix networks. La risposta del pacchetto segue lo stesso criterio sfruttando nodi, algoritmi e chiavi differenti. Questo concede all'utilizzatore di rimanere completamente anonimo, cosa che non può essere avvenire con la semplice criptografia SSL che agisce esclusivamente sul corpo del messaggio
Come prima operazione per utilizzare la rete onion il client deve generare un nuovo circuito definendo il percorso di nodi che ogni pacchetto dovrà seguire, iniziando dal primo nodo vengono scambiate informazioni crittografiche come l'algoritmo e le chiavi da usare, poi si usano le informazioni finora ottenute per ottenere quelle del prossimo nodo e cosi via fino a che non si è generato tutto il circuito, grazie a questo meccanismo neanche durante la generazione del circuito è possibile risalire al client. Viene usata la crittografia asimmetrica per scambiare le chiavi simmetriche tra il client e ogni router, questo viene fatto in quanto la crittografia asimmetrica è molto costosa e viene quindi usata solo alla generazione del circuito, successivamente gli strati vengono criptati e decriptati con la stessa chiave simmetrica. Questo consente alla rete onion ad avere una bassa latenza che è incrementata solo dal numero di onion router nel percorso e non dalla tecnologia che non è distante da quella usata in HTTPS [GD99]

2.1 Applicazioni di utilizzo

L'onion routing può essere usato con una moltitudine di protocolli e applicazioni, tra i più comuni troviamo HTTP(S), FTP, SSH, SMTP, DNS e VPNs. L'utilizzo di molti dei protocolli più comuni avviene tramite gli onion proxies, i quali sono suddivisi in tre proxy layer logici

- Un proxy che genera e gestisce le connessioni, per operare ha necessità di conoscere la topologia e i percorsi verso altri nodi, tutte le informazioni vengono distribuite in modo sicuro all'interno della rete a ogni nuovo nodo che si connette
- Un proxy chiamato ***"Application Specific Proxy"***, a una connessione la relativa applicazione invia al proxy il pacchetto che normalmente invierebbe al server di destinazione, il proxy si occupa di convertire lo stream di dati in un formato accettato dalla rete onion
- Un proxy opzionale chiamato ***"Application Specific Privacy Filter"*** che sanifica lo stream di dati rimuovendo informazioni che potrebbero identificare la sorgente

[GD99]
I proxy possono essere configurati in molteplici modi, tra i quali vi è la possibilità di eseguire il proxy in un server remoto e sfruttare la rete Tor senza dover installare il software in ogni dispositivo, che quindi non ne deve gestire la computazione

2.2 Onion come Mix Network

La rete Tor è una delle molteplici reti basate sullo studio di David Chaum sulle Mix network, il suo studio si concentrò sulla rete basata sulla crittografia asimmetrica e sui sistemi di risposta senza conoscere l'identità del destinatario. Tutta la rete doveva funzionare senza necessità di un'ente centrale a gestire le connessioni.
In particolare abbiamo chiave pubblica K e chiave privata P, abbiamo inoltre

- K(x) la funzione che cripta x con la chiave pubblica, può solo essere decriptato con la chiave P
- P(x) la funzione che cripta x con la chiave privata, può solo essere decriptato con la chiave K

Abbiamo quindi $P(K(x)) = K(P(x)) = x$
Con questa tecnica però qualcuno potrebbe determinare il contenuto di un messaggio creando una copia identica in quanto $y = x ß K(y) = K(x)$, per risolvere questo problema si esegue la crittografia del messaggio inserendo una stringa casuale R ottenendo quindi chiavi sempre diverse tramite le funzioni $K(x, R)$ e $P(x, R)$, questa tecnica è chiamata sealing
In una rete questo sistema viene implementato in maniera ridondante, il messaggio M viene criptato insieme a una stringa casuale R con la chiave pubblica del destinatario Kb, il tutto viene criptato assieme all'indirizzo B e a una stringa casuale R1 con la chiave pubblica K1 del nodo 1, questo processo potrebbe essere esteso con N nodi (mix) rendendo quindi la rete molto sicura $K1(R1, Kb(R0, M), B)$
Quando il nodo 1 riceve il pacchetto lo decripta con la sua chiave privata scartando R1 ottiene $Ka(R0, M), B$. Inoltra quindi il nuovo pacchetto a B
Il destinatario di un pacchetto deve avere la possibilità di rispondere senza conoscere l'indirizzo di A.
A sfrutta il proprio indirizzo reale per generare un indirizzo non tracciabile, in particolare genera due chiavi pubbliche Ka e R1, cripta il proprio indirizzo A assieme alla chiave R1 come stringa casuale usando la chiave del mix
$K1(R1, A), Ka$
Quando B deve rispondere al pacchetto usa Ka per criptare il messaggio e $K1(R1, A)$ come indirizzo di destinazione, il mix M1 che riceve il pacchetto usa la chiave privata P1 per decriptare l'indirizzo di destinazione (A) ed usa la stringa casuale R1 come ulteriore chiave per criptare il messaggio
$K1(R1, A), Ka(R0, M) => A, R1(Ka(R0, M))$
Con questo sistema B può rispondere ad A, non conoscendo il vero indirizzo di A, il mix non conosce il contenuto del messaggio ma conosce l'indirizzo di destinazione e A è l'unico che può decriptare il contenuto del messaggio
Da considerare che M1 è il primo mix nel percorso da A a B, mentre tra M1 e B possono esserci N nodi ed il messaggio può quindi essere criptato più volte nel percorso da B a M1
M1 viene quindi considerato un nodo di uscita dalla rete dato che è l'unico che conosce il vero indirizzo di destinazione
Nelle Mix Networks abbiamo un ente centrale che possiede una lista di pseudonimi creati dagli utenti. Quando un utente vuole creare il proprio pseudonimo invia la richiesta all'ente che decide se accettare la richiesta, solo allora lo aggiunge alla lista.
La richiesta contiene una chiave pubblica che agisce da pseudonimo, viene usata per verificare le firme generate dall'utente tramite la relativa chiave privata Le richieste

all'ente possono essere inviate in maniera anonima, sfruttando un email non tracciabile o un'altro sistema.

[Cha81]

3. Tor § Onion v2

Nel 2002 viene presentata la rete Tor, diventata open source 2 anni dopo è l'implementazione più famosa di onion routing che porta alcuni miglioramenti sostanziali
Migliore segretezza del canale, nella versione originale un nodo poteva forzare altri onion router nel circuito a decriptare il traffico precedentemente registrato. La rete tor sfrutta una tecnica di circuiti telescopici in cui il client che genera il circuito crea chiavi di sessione di breve durata che quindi non potranno essere usate dai malintenzionati per decriptare il vecchio traffico
L'implementazione del proxy di applicazione attraverso lo standard SOCKS, consente alla maggior parte del traffico TCP di funzionare senza modifiche. Precedentemente era necessario implementare un proxy per ogni applicazione, era quindi necessario generare un circuito per ogni applicazione, con conseguente duplicazione di chiavi, in Tor invece il circuito viene generato a livello di TCP e più applicazioni possono sfruttarlo.
Per garantire la non tracciabilità di un utente nello stesso stream dati usato da più applicazioni è stato implementato il meccanismo dei rotating circuits che genera ogni minuto un nuovo circuito se quello precedente non viene usato

- Controllo di congestione, un sistema decentralizzato che sfrutta ack end-to-end per garantire l'anonimato
- Directory Server, nodi più fidati di altri che descrivono le informazioni di rete in maniera sicura e affidabile
- Politiche di uscita variabili, ogni nodo possiede delle politiche che specificano le connessioni consentite e rifiutate, fondamentale in una rete distribuita fatta da volontari
- Controllo di integrità end-to-end, viene eseguito un controllo di integrità nel momento in cui il pacchetto esce dalla rete per garantire che i contenuti non sono stati alterati

Tor, a differenza degli altri sistemi che implementano le Mix-Network di Chaum predilige la bassa latenza, il che rende la rete adatta all'utilizzo tramite un web browser, questo inoltre migliora l'usabilità di tor il che è un aspetto fondamentale dato che maggiori sono i nodi e più semplice garantire l'anonimato.
Tor non è completamente sicuro, infatti non filtra informazioni di privacy nel corpo del messaggio come invece avviene in altri sistemi come Privoxy o Anonymizer e non offre garanzie in caso di attacco end-to-end che concerne sia sorgente che destinazione [RDos]

3.1 Obiettivi

L'obiettivo principale della rete TOR è la creazione di una rete che possa rendere gli attacchi molto più complessi da portare a termine scoraggiando cosi ogni possibile hacker, da questo principio cardine sono derivati gli altri obiettivi, tra cui

- Usabilità, ogni sistema che implementa le Mix Networks ha bisogno di utenti e quindi nodi per rendere la rete sicura. L'usabilità garantisce la sicurezza. Da questo derivano altri obiettivi come
  - Basse latenze, determinate anche dal fatto che più applicazioni possono usare lo stesso circuito TCP senza doverne generare uno nuovo per ogni stream dati, questo consente di ridurre il delay causato dalla criptografia asimmetrica
  - Essere implementabile con meno configurazioni possibili
  - Essere multi piattaforma
- Semplicità, la rete deve essere facile da comprendere, doveva essere usabile nel mondo reale e non doveva essere troppo costosa

[Cha81]

3.2 Network Design

A differenza di altri sistemi che usano una rete peer-to-peer, Tor è una *overlay network*, questa scelta è derivata dalle possibili vulnerabilità di una rete basata sugli utenti, in particolare un attaccante potrebbe compromettere il traffico leggendo o manipolando i dati.
Una *overlay network* è una rete virtuale creata sfruttando una rete fisica preesistente, infatti ogni onion router è rappresentato come un processo software che mantiene due tipi di chiavi

- Chiave d'identità, una chiave di lunga durata usata per firmare i pacchetti garantendo agli altri nodi della rete l'autenticità del messaggio e delle informazioni contenute, tra cui indirizzo, bandwidth, exit policy ecc
- Chiave onion, una chiave di breve durata usata per decriptare le richieste all'interno dei circuiti utente

Un elemento chiave della rete TOR sono le celle, tutto il traffico della rete passa attraverso pacchetti di dimensione fissa 512 bytes chiamati celle, come ogni tipo di pacchetto sono divisi in header e payload, l'header contiene l'identificativo del circuito e un comando che indica come gestire il payload. Il comando può essere

- Padding per mantenere viva la connessione
- Create per creare un nuovo circuito
- Destroy per eliminare un circuito

Inoltre il tipo di comando definisce il tipo della cella

- Control, non vengono inoltrati ma sono gestiti dal primo router che li riceve
- Relay, trasporta stream dati per cui ha necessità di un header aggiuntivo contenente lo streamID, checksum per il controllo di integrità, la dimensione del payload e un comando di relay
- Il comando di relay può essere
- Begin per iniziare uno stream
- End per terminare uno stream
- Teardown per terminare uno stream in modo forzato, usato per quelli rotti
- Connected per rispondere al relay begin dell'OP, informandolo che lo stream è stato creato con successo
- Extend per estendere un circuito di un router
- Truncate per eseguire un teardown di una parte del circuito
- Sendme usato per il controllo di congestione
- Drop

L'utente per generare il circuito segue un processo di negoziazione incrementale:

1. Come primo passo l'utente, in particolare l'Onion Proxy invia una richiesta relay create al primo nodo nel percorso scelto
2. Avviene la condivisione della chiave simmetrica tramite l'handshake di Diffie-Hellman, la creazione dell'ID del circuito e di conseguenza la creazione della connessione con il primo nodo (R1)
3. L'utente invia una richiesta relay extend indicando a R1 l'indirizzo del secondo nodo nel percorso scelto (R2), R1 inizia una connessione con R2 definendo un nuovo id e associando la connessione OP-R1 con la connessione R1-R2, OP e R2 non si conoscono a vicenda e comunicano solo tramite l'intermediario R1. In fine R1 invia all'utente le informazioni del nuovo nodo tra cui la chiave simmetrica per il relativo strato
4. Questa operazione viene effettuata, richiedendo all'ultimo router corrente di creare una connessione con il suo successivo, finché il circuito non viene completato[Cha81]

Ogni applicazione, in base alle proprie necessità, invia le richieste di stream TCP all'Onion Proxy, che sceglie il circuito più recente (oppure genera un nuovo circuito) e sceglie un exit node, solitamente l'ultimo router. A questo punto l'OP invia una cella *relay begin* con un identificativo casuale all'exit node, la risposta dell'exit node conferma l'esistenza del nuovo stream TCP e l'OP può accettare i dati delle applicazioni TCP ed inoltrarli nella rete Onion
Questo design ha qualche problema derivato dal fatto che più OP potrebbero scegliere lo stesso percorso OR-OR, generando un bottleneck e saturando la rete. Le normali

tecnologie di controllo di congestione non possono funzionare in una rete del genere, i pacchetti devono rispettare il proprio percorso e non possono cambiarlo, questo ne renderebbe impossibile la lettura. Per implementare un sistema di controllo di congestione sono stati implementati 2 meccanismi

- **Controllo di Circuito**, ogni OP possiede le informazioni di entrambe le finestre per ogni OR, mentre ogni OR possiede le informazioni delle finestre del relativo OP per ogni circuito. Le finestre iniziano da un valore di 1000 e decrementano a ogni cella, sono
  - **Packaging window**, tiene traccia del numero di celle un OR può inviare verso l'OP, quando un router riceve abbastanza celle dati (100) invia un relay sendme all'OP con $streamID = 0$, il quale incrementa la finestra per il relativo OR, se la finestra raggiunge 0 il router smette di ricevere celle dal circuito attendendo il sendme. Lo stesso meccanismo vale per l'OP
  - **Delivery window**, tiene traccia del numero di celle un OR è in grado di inviare fuori dalla rete
- **Controllo di Stream**, simile al controllo di circuito ma il meccanismo è applicato ad ogni stream TCP separatamente. Ogni stream inizia con packaging = 500 con incrementi di 50 ad ogni relay sendme

3.3 Servizi Onion

Nelle normali reti internet i servizi vengono utilizzati sulla base del relativo indirizzo IP, la posizione è conosciuta da tutti. Onion invece garantisce l'anonimato non solo ai clienti ma anche a chi mette a disposizione server per fornire servizi, nascondendo l'indirizzo IP.
Questo viene fatto tramite uno pseudonimo di lungo termine, identico in tutti i circuiti e stabile anche ad un fallimento di un router
I principali obiettivi sono

- Gli attaccanti non devono riuscire a manipolare la rete sostituendosi un servizio esistente. Questo livello di affidabilità deriva dalla criptografia asimmetrica che ci garantisce che il servizio a cui cerchiamo di connetterci sia autentico, solo lui possiede la copia privata della chiave pubblica con cui tentiamo la connessione

- Sicurezza dagli attacchi DoS, viene fatto tramite l'uso di più punti d'ingresso alla rete

La creazione di un servizio onion passa per diversi punti prima di poter essere raggiungibile

1. Genera una coppia di chiave pubblica e privata per identificarsi

2. Definisce alcuni onion router come punti di ingresso nella rete, da cui riceverà le richieste dei clienti e invia loro la chiave pubblica

3. Crea un circuito con ogni punto di ingresso

4. Invia al servizio di lookup onion le informazioni sui punti d'ingresso e l'hash della chiave pubblica che sarà usata come hostname

Quando un utente tenta la connessione

1. Inizialmente esegue il lookup del dominio

2. Successivamente sceglie un OR come tramite per il servizio e genera un circuito verso di esso, sfruttando uno specifico cookie per identificare il servizio

3. Viene generato uno stream verso uno dei punti d'ingresso del servizio con tutte le informazioni riguardo se stesso, il nodo tramite e il cookie. Tutto viene criptato con la chiave pubblica del servizio

4. Inizia l'handshake di Diffle-Hellman tra OP e servizio

5. Il servizio onion a sua volta genera un circuito verso il nodo tramite per poter rispondere all'OP con la seconda parte dell'handshake

6. Il nodo tramite collega i due circuiti generando un circuito dati bidirezionale

Sia l'utente che il server non vengono modificati e il server non è nemmeno a corrente che il suo traffico viaggia per una rete onion [Cha81]

3.4 Directory Servers

I directory server sono un piccolo sottogruppo di onion router utilizzati per tracciare i cambiamenti nella topologia di rete. In particolare un directory server agisce come un server HTTP accessibile dai client per ottenere lo stato della rete e la lista dei router aggiornata periodicamente dagli stessi OR. Il software di accesso alla rete onion è precaricato con le informazioni sui directory server e le relative chiavi
Quando il directory server riceve un aggiornamento da un OR prima di tutto controlla la chiave d'identità del router, garantendo che un attaccante non possa fingersi un OR manomettendo la rete [Cha81]

3.5 Vulnerabilità

3.6 Tor Browser

4. Implementazione

4.1 Onion V3

La terza generazione di onion nasce per risolvere alcuni problemi di sicurezza. In particolare rispetto alla seconda generazione

- Vengono aggiornati i sistemi di crittografia, da SHA1/DH/RSA1024 a SHA3/ed25519/curve25519
- Vengono migliorati i directory server e il directory protocol
- Viene cambiato il sistema di hostname

Precedentemente venivano usati i primi 80 bit dell'hash (SHA1) della chiave pubblica per creare un'hostname, un esempio yyhws9optuwiwsns.onion, nella terza generazione vengono codificati in base32

- La chiave pubblica, un totale di 32 byte in ed25519
- Il checksum di 2 byte
- Un byte di versione, di default '\x03'

In totale il nuovo hostname possiede 56 caratteri [Tor13]
Un esempio è l'indirizzo *pg6mmjiyjmcrsslvykfwnntlaru7p5svn6y2ymmju6nubxndf4pscryd.onion*, una volta decriptato sfruttando il base32 definito nell' RFC 3548 e 4648 abbiamo la seguente stringa 79bcc625184b05194975c28b66b66b0469f7f6556fb1ac3189a79b40dda32f1f214703, come notiamo termina con 03, il byte di versione

4.2 Studio delle tecnologie

Per creare un servizio sulla rete onion si possono usare una moltitudine di tecnologie differenti, ogni singolo aspetto necessita di effettuare delle scelte.

- Deploy del Server, la prima scelta ricade sulla creazione del server e in particolare scegliere dove eseguire il deploy, ci sono molteplici motivi per usare un servizio cloud, tra cui la sicurezza e la facilità d'uso. Ci sono 3 principali competitor e molti altri secondari che per la maggior parte sfruttano i server di queste 3 aziende
  - Microsoft Azure
  - Google Cloud

  - Amazon Web Service, il più importante servizio cloud, è gestito e reso disponibile da Amazon e possiede molti servizi tra cui scegliere. Per questa implementazione useremo il servizio EC2 messo a disposizione da AWS in una macchina t3.micro

- SO, la seconda scelta ricade sul sistema operativo della nostra macchina. Debian in particolare ha molti vantaggi, tra cui

  - Leggerezza, affidabilità e sicurezza derivate da un sistema GNU/Linux
  - Maggiore utilizzo in ambienti server
  - Nessun costo aggiuntivo derivato dall'aquisto di una licenza d'uso come Windows server

  Useremo in particolare la versione 11

- Web Server, abbiamo due web server principali

  - Apache httpd, il più vecchio dei due, rilasciato con licenza Apache 2.0
  - Nginx, un web server più moderno che fornisce diversi altri strumenti oltre al web server, tra cui il proxy e load balancer, è gratuito e come httpd è open source, ha superato apache da un paio di anni. Viene anche utilizzato in container docker, ma noi lo useremo in una classica macchina virtuale.

4.3 Creazione del servizio

Per questa implementazione verrà usato il servizio di macchina virtuale (EC2) con Debian 11 messo da Amazon Web Service tramite il web server nginx
Il primissimo passo fondamentale è aggiungere alla macchina i giusti security group, altrimenti la connessione non sarebbe possibile

- Consentire la connessione ssh alla porta 22

Type	Protocol	Port range	Source
SSH	TCP	22	0.0.0.0/0

Figura 4.1: security Group 1, ssh in entrata

- Consentire il traffico TCP in uscita

IP version	Type	Protocol	Port range	Destination
IPv4	All TCP	TCP	0 - 65535	0.0.0.0/0

Figura 4.2: security Group 2, TCP in uscita

È inoltre importante creare una coppia chiave pubblica/privata, useremo la chiave privata con formato *.pem per tentare la connessione

```
ssh −i key.pem admin@∗.compute.amazonaws.com
```

La primissima operazione sarà l'aggiornamento dei repository e successivamente del sistema

```
sudo apt update && sudo apt full−upgrade −y
```

Poi installiamo il web server

```
sudo apt install nginx
```

Una volta completata l'installazione il web server è già avviato e in ascolto sulla porta 80

4.4 Tor & unix socket

Per configurare tor è innanzitutto necessario aggiungere il repository, installiamo **apt-transport-https** per utilizzare i repository tramite https

```
sudo apt install apt−transport−https
```

Poi aggiungiamo i repository nella cartella /etc/apt/sources.list.d
Dal comando lsb_release −c possiamo vedere la distribuzione corrente e inserirla nella configurazione, in questo caso bullseye è la nostra distribuzione, inseriamo nel file tor.list i seguenti

```
deb [signed−by=/usr/share/keyrings/tor−archive−keyring.gpg] https://deb.
    torproject.org/torproject.org bullseye main
deb−src [signed−by=/usr/share/keyrings/tor−archive−keyring.gpg] https://deb.
    torproject.org/torproject.org bullseye main
```

Aggiungiamo le chiavi gpg della repository appena aggiunta

```
wget −qO− https://deb.torproject.org/torproject.org/
    A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89.asc | gpg −−
    dearmor | tee /usr/share/keyrings/tor−archive−keyring.gpg >/dev/null
```

Assicuriamoci inoltre che gpg sia installato nel sistema

```
sudo apt install gpg
```

Aggiorniamo gli index ed installiamo finalmente tor

```
sudo apt update && sudo apt install tor −y
```

[Tora] Generiamo il torrc file che ci servirà per impostare tutti i parametri di configurazione

```
sudo tor −f /etc/tor/torrc
```

Apriamo il file e togliamo il commento alle righe HiddenServicePort 80 127.0.0.1:80 e HiddenServiceDir ∗, il primo ci serve per indicare la directory in cui si trovano le informazioni del sito e le chiavi crittografiche, il secondo indica che il traffico arrivato dalla porta virtuale (80), che gli utenti onion useranno per la connessione, viene reindirizzato alla porta 80 del localhost, ovvero la porta in cui è in ascolto il web server nginx.
Riavviamo tor per assicurarci che il file di configurazione torrc non abbia errori

```
sudo systemctl restart tor
```

A questo punto tor ha creato gli introduction points e ha generato un circuito con ognuno di essi, ha generato le chiavi ed il proprio hostname, queste informazioni sono state aggiunte nella cartella che abbiamo inserito nell'HiddenServicePort, in particolare hostname contiene l'indirizzo tor del nostro servizio [Torb]
Questo sistema però non è completamente sicuro, stiamo collegando tor con il web server tramite una porta, essendo entrambi i processi sulla stessa macchina il sistema ottimale è utilizzare un socket unix tra i due processi. Questo impedisce ad un utente non Tor di accedere direttamente al web server senza dover configurare firewall che comunque aggiungono complessità nella rete
Apriamo il file /etc/nginx/sites−enabled/default e nella sezione server aggiungiamo il socket in ascolto, cosi che la comunicazione possa passare anche per il socket unix

```
listen unix:/var/run/website.sock;
```

Possiamo inoltre rimuovere la connessione dalla porta 80 commentando le seguenti righe

```
#listen 80 default_server;
#listen [::]:80 default_server;
```

Dopo aver riavviato nginx ed esserci assicurati che non vi siano errori apriamo il file torrc e inseriamo lo stesso socket

```
HiddenServicePort 80 unix:/var/run/website.sock
```

Infine riavviamo tor, se tutto è andato a buon fine il servizio dovrebbe essere in grado di rispondere

```
sudo systemctl restart tor
```

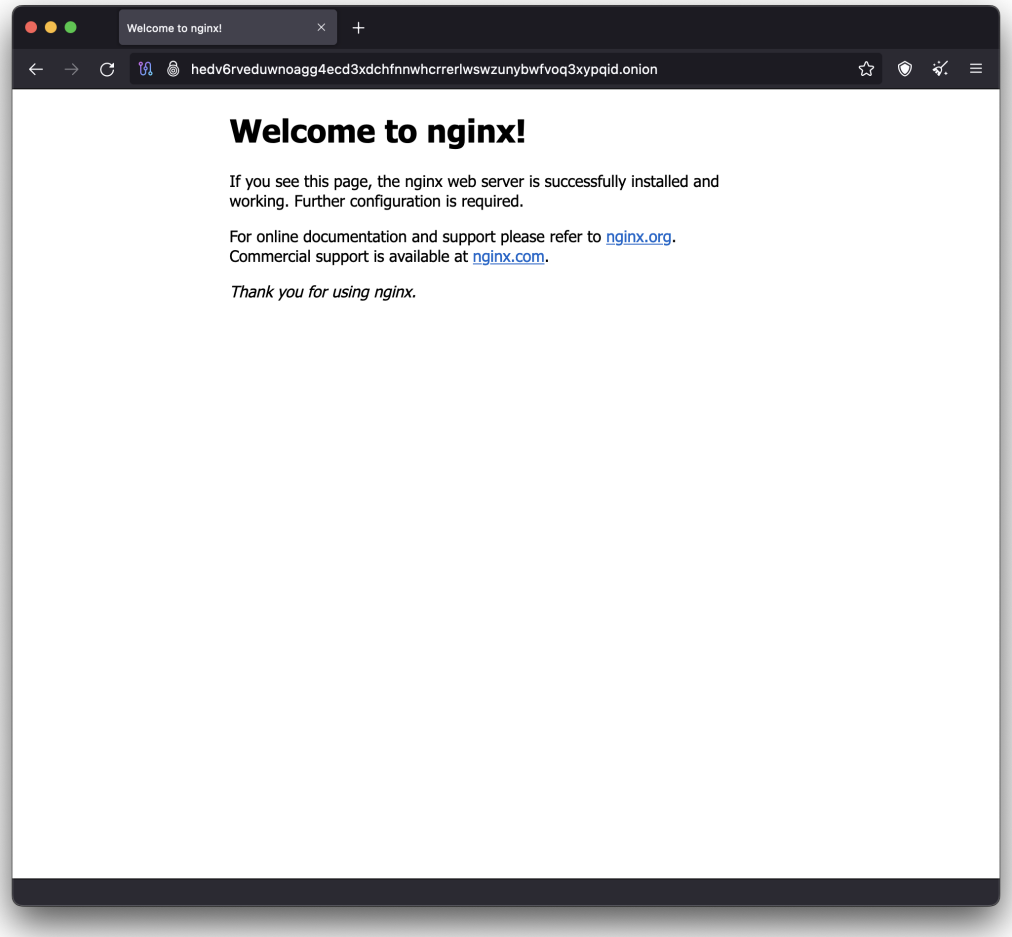

Figura 4.3: Connessione al web server nginx

5. Capitolo Esempio

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed do eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullamco laboriosam, nisi ut aliquid ex ea commodi consequatur. Duis aute irure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

In questo capitolo andremo a discutere ...

5.1 Sezione Esempio

Quello in Figura **??** (esempio di riferimento a figura) ...

Esempio elenco puntato ...

- item 1
- item 2
- item 3

5.2 Section2

5.2.1 Subsection Esempio

5.2.2 Subsection Esempio

```
1       GET /chat HTTP/1.1
2       Host: server.example.com
3       Upgrade: websocket
4       Connection: Upgrade
5       Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
6       Origin: http://example.com
7       Sec-WebSocket-Protocol: chat, superchat
8       Sec-WebSocket-Version: 13
```

Listing 5.1: Esempio di listing

Nelle Tabelle 5.1 e 5.2 è possibile vedere, rispettivamente per desktop e per mobile, il supporto dei vari browser per le diverse specifiche delle WebSocket. Il codice completo dell'esempio è disponibile sul mio GitHub[1] (Esempio di link).

[1] https://github.com/Glydric/TesiTriennale

Versione	Chrome	Firefox	Internet Explorer	Opera	Safari
76	6	4.0	No	11.00(disabilitato)	5.0.1
7	No	6.0	No	No	No
10	14	7.0	HTML5 Labs	?	?
RFC 6455	16	11.0	10	12.10	6.0

Tabella 5.1: Esempio di Tabella

Versione	Android	Firefox Mob.	IE Mob.	Opera Mob.	Safari Mob.
76	?	?	?	?	?
7	?	?	?	?	?
10	?	7.0	?	?	?
RFC 6455	16(Chrome)	11.0	?	12.10	6.0

Tabella 5.2: Esempio di Tabella

5.3 Qui

Ciao Ciao

Ciao Ciao
Ciao Ciao

Bibliografia

[Cha81] David Chaum. ≪Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms≫. In: *ACM* (1981).

[GD99] Syverson P. Goldschlag D. Reed M. ≪Onion Routing for Anonymous and Private Internet Connections≫. In: (1999).

[RDos] Paul Syverson Roger Dingledine Nick Mathewson. ≪Tor: The Second-Generation Onion Router≫. In: (agosto 2004).

[Tora] Tor. *enable official repo*. URL: https://support.torproject.org/apt/tor-deb-repo/.

[Torb] Tor. *setup onion service*. URL: https://community.torproject.org/onion-services/setup/.

[Tor13] Tor. *Rendezvous Specification - Version 3*. 2013. URL: https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt.

Ringraziamenti

Ringrazio...