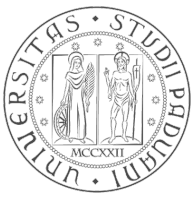

UNIVERSITÀ DEGLI STUDI DI PADOVA

SCUOLA DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

ATTACCHI DI TIPO DENIAL OF SERVICE SULLA RETE DI ANONIMATO TOR

Relatore: Ch.mo Prof. NICOLA LAURENTI

Laureando: NICOLA MARINELLO

Matricola 1071102

21 Settembre 2016

ANNO ACCADEMICO 2015-2016

Abstract

Il tema della segretezza nelle comunicazioni è diventato sempre più di maggiore rilievo negli ultimi anni. Intercettazioni, limitazione della libertà di espressione, censura da parte dei governi e in generale tutto ciò che limita lo scambio di informazioni tra entità hanno portato ad un crescente interesse verso le comunicazioni protette. Per questi motivi sono stati sviluppati numerosi sistemi che permettono comunicazioni in forma anonima e cifrata, tra questi Tor. È importante comprendere che anche se Tor sta diventando sempre più popolare, i suoi utenti sono potenziali obiettivi di attacchi che mirano ad identificarli o impedirne l'utilizzo. In questa tesi verranno discussi i principali attacchi di tipo *Denial of Service* (DoS) che possono essere condotti nella rete di anonimato Tor. Nella prima parte verranno introdotte delle nozioni basilari di crittografia, verrà spiegato cos'è la rete Tor e come funziona. Successivamente verrà fatta una panoramica sugli attacchi e su possibili contromisure per ognuno.

Indice

Introduzione

Ogni volta che inviamo una e-mail, visitiamo un sito web o chattiamo con qualcuno, i nostri pacchetti attraversano vari router/server che possono controllare i dati inoltrati. Anche se i dati contenuti nei pacchetti sono crittografati, l'IP header rimane comunque visibile, ed è quindi possibile scoprire le identità del mittente e del destinatario. Intercettando e analizzando i pacchetti, una spia può ottenere un numero considerevole di informazioni circa l'identità dei soggetti della comunicazione, l'applicazione che li ha generati e talvolta il loro contenuto. I sistemi di anonymous routing, come l'*Onion Routing*[1], nascono per garantire la segretezza nelle comunicazioni e l'anonimato delle parti coinvolte.

L'Onion routing è il più diffuso sistema di comunicazione anonimo a bassa latenza, permette web browsing, invio di e-mail, messaggistica istantanea e altri servizi. Esso si basa sulla rete *Tor (The Onion Router)*, formata da un gruppo di volontari, che donano la propria banda per permettere agli utenti di aumentare la loro privacy e la loro sicurezza. La rete viene anche utilizzata come strumento per aggirare censure e blocchi imposti dagli ISP e, più in generale, il controllo delle comunicazioni da parte dei governi e dei regimi repressivi. Milioni di persone ogni giorno usano Tor per svolgere le loro attività quotidiane (come e-mail, Facebook, Twitter) senza il timore di essere monitorati. Per questo in alcuni paesi del mondo, come Cina, Iran, Kazakistan ecc. si tenta di arginare il più possibile la diffusione e l'utilizzo di Tor. In altri paesi, come gli Stati Uniti, viene usato dalla Marina Militare per svolgere operazioni di intelligence e dalle forze dell'ordine per controllare siti web, senza lasciare tracce di indirizzi IP governativi. Anche organizzazioni come Wikileaks utilizzano Tor per scambiare informazioni e tutelare i propri informatori. Attualmente, la rete conta circa 7000 router attivi e approssimativamente 2 milioni di client che si collegano ogni giorno [2].

Attacchi DoS

L'acronimo *DoS*, abbreviazione di *denial of service*, letteralmente negazione del servizio, indica una tipologia di attacchi informatici che mirano ad esaurire le risorse di un sistema, tipicamente un web server o un router, fino a renderlo non più in grado di fornire i propri servizi ai client. Esso non è un attacco caratteristico della rete Tor, dato che, attraverso varie tecniche, può essere effettuato contro qualsiasi sistema collegato ad Internet che offre servizi TCP. Quello che è possibile fare, però, è sfruttare delle debolezze di progettazione del protocollo di Tor per condurre degli attacchi che in una normale rete TCP non sarebbero possibili. Come nei classici attacchi DoS, anche quelli che sfruttano le vulnerabilità di Tor mirano al consumo di banda, di risorse computazionali o di memoria. Gli attacchi DoS sulle reti di anonimato possono essere suddivisi in due categorie: *blanket blocking*, che blocca l'accesso all'intera rete senza attaccare in modo diretto alcun router, come ad esempio i blocchi imposti dai governi, oppure *targeted attack*, ovvero attacchi con un obiettivo specifico, che tentano di portare nodi della rete offline. Portando offline i router portanti della rete, è possibile, probabilisticamente parlando, deanonimizzare servizi nascosti, scoprire l'identità di due interlocutori o comunque causare un calo di prestazioni della rete che può indurre gli utenti ad utilizzare altri metodi di comunicazione meno sicuri di Tor.

1 Background

Nelle sezioni successive verranno presentate delle nozioni basilari di crittografia e di protocolli di sicurezza necessari a comprendere la seconda parte della tesi. In particolare verranno presentati gli algoritmi crittografici e i protocolli utilizzati negli scambi di dati attraverso la rete Tor e i metodi con cui vengono stabilite le connessioni sicure.

1.1 Crittografia simmetrica e asimmetrica

La *crittografia* è la branca della crittologia che studia i metodi per rendere un messaggio "offuscato", in modo da non essere comprensibile a persone che non sono autorizzate a leggerlo. Uno scambio di messaggi crittografati tra due interlocutori è costituito da 5 elementi principali:

- *messaggio originale* $\mathcal{P}$ in chiaro che si vuole trasmettere;
- *chiave crittografica* $\mathcal{K}_1$, indipendente dal messaggio;
- *algoritmo di crittografia* $\mathcal{F}_C$, che si occupa di cifrare il messaggio attraverso la chiave crittografica $\mathcal{K}_1$;
- *messaggio cifrato* $C = \mathcal{F}_C(\mathcal{P}, \mathcal{K}_1)$, ottenuto da un algoritmo crittografico che riceve in input un messaggio in chiaro e una chiave ;
- *algoritmo di decrittografia* $\mathcal{F}_D$ che si occupa di ritornare al messaggio originale a partire da quello cifrato da una chiave $\mathcal{K}_2$ ($\mathcal{P} = \mathcal{F}_D(C, \mathcal{K}_2)$).

Con questi elementi a disposizione Alice (A) e Bob (B) ad esempio, possono comunicare in maniera sicura, senza che i loro messaggi, eventualmente intercettati da una spia (E), possano essere decifrati. Possiamo inoltre distinguere due categorie di tecniche crittografiche: tecniche a chiave *simmetrica* e *asimmetrica*.

Tecniche a chiave simmetrica

Nelle tecniche a chiave simmetrica (o a chiave privata) la chiave usata per cifrare il messaggio e per decifrarlo sono uguali ($\mathcal{K}_1 = \mathcal{K}_2$). Gli algoritmi che fanno uso di questa tecnica sono molto performanti ma sia A che B devono essere in possesso della chiave, che non può essere scambiata attraverso un canale insicuro. Lo scambio infatti può avvenire tramite algoritmi asimmetrici, che descriveremo tra poco. Un altro parametro molto importante che influisce fortemente sulla sicurezza del cifrario è la lunghezza n della chiave, che generalmente è di 128 o 256 bit. Una chiave troppo corta da la possibilità ad una spia di provare tutte le possibili combinazioni (2^n) e indovinare quella giusta, così da decifrare il messaggio, mentre se è sufficientemente lunga occorre una quantità di risorse troppo elevata per poter essere provato nella pratica. Gli algoritmi più famosi che implementano questa tecnica sono AES, DES, 3DES ecc.

Tecniche a chiave asimmetrica

Al contrario delle tecniche a chiave simmetrica, in quelle di tipo asimmetrico (conosciute anche come tecniche a chiave pubblica) le chiavi usate per cifrare e decifrare il messaggio sono diverse ($\mathcal{K}_1 \neq \mathcal{K}_2$). Due interlocutori, A e B che vogliono comunicare in modo sicuro, non hanno necessità di scambiarsi una chiave privata da utilizzare per cifrare i messaggi: B distribuisce pubblicamente una chiave $\mathcal{K}_1$ che *chiunque* può utilizzare per cifrare dei

messaggi da inviargli. Solo B sarà poi in grado di decifrare i messaggi attraverso la chiave privata $\mathcal{K}_2$ in suo possesso. La forza di un sistema di questo tipo si basa sul fatto che a partire dalla chiave pubblica è molto difficile riuscire a scoprire la chiave privata. Questo è teoricamente possibile, ma in pratica non esistono dei metodi efficienti per farlo; infatti qualsiasi computer è in grado di moltiplicare due numeri primi di 150 cifre in pochissimo tempo, ma fattorizzare il risultato per trovare i due numeri di partenza risulta un'operazione troppo dispendiosa. Essendo più lenti di quelli a chiave simmetrica, gli algoritmi che implementano queste tecniche molto spesso vengono utilizzati solamente per scambiare una chiave simmetrica (detta anche chiave di sessione) da utilizzare durante tutto il resto della comunicazione. Questa fase è detta negoziazione della chiave. Algoritmi famosi a chiave asimmetrica sono: RSA, Diffie-Hellman, Digital Signature Algortithm ecc.

1.2 Algoritmi e protocolli crittografici usati in Tor

Come indicato nelle specifiche di Tor [3], vengono utilizzati più algoritmi sia a chiave simmetrica che asimmetrica per la cifratura dei dati scambiati. I due interlocutori al momento della connessione scambiano una ciphersuite (un elenco dei metodi crittografici supportati da entrambi) e da questa scelgono quali algoritmi e protocolli utilizzare, anche se qui verranno trattati quelli usati nella maggioranza dei casi.

RSA

RSA è un algoritmo di crittografia a chiave pubblica, utilizzato nel protocollo Tor solo al momento della connessione con un router. Ogni Onion Router è in possesso di una chiave, detta *Onion key* a 1024 bit che distribuisce pubblicamente; chiunque voglia stabilire una connessione con esso, deve inizialmente comunicare attraverso questa chiave. Essa è una chiave a medio termine, dato che viene cambiata periodicamente.

AES

AES è un algoritmo crittografico a chiave simmetrica utilizzato in Tor per la cifratura della quasi totalità dei dati scambiati nella rete. Essendo la chiave segreta, essa viene scambiata attraverso il protocollo di Diffie-Hellman che verrà discusso tra poco. Una volta scambiata, viene usata per tutta la durata della comunicazione, da qui il nome di *chiave di sessione*.

Diffie-Hellman

Lo scambio di chiavi Diffie-Hellman, conosciuto anche come D-H key exchange, è un protocollo crittografico che consente a due entità di concordare una chiave condivisa segreta utilizzando un canale di comunicazione insicuro. La chiave ottenuta dopo lo scambio viene utilizzata per cifrare tutto il resto della comunicazione con degli algoritmi di crittografia simmetrica. Si considerano inizialmente due numeri g e p concordati precedentemente tra le parti (come nel caso di Tor visto che sono fissati dalle specifiche) o scelti all'inizio della comunicazione. Uno dei due interlocutori, ad esempio Alice, sceglie un numero casuale a e calcola il valore $A = g^a \mod (p)$ (dove $\mod(\cdot)$ indica il resto della divisione intera) e lo invia attraverso il canale insicuro a Bob, eventualmente insieme a g e p se non erano stati concordati. Anche Bob sceglie un numero casuale b e calcola il valore $B = g^b \mod (p)$ e lo invia ad Alice. A questo punto Alice calcola $K_A = B^a \mod (p)$ mentre Bob calcola $K_B = A^b \mod (p)$. I valori K_A e K_B sono uguali, quindi i due interlocutori sono in possesso di una chiave segreta che possono iniziare ad utilizzare. La forza di questo protocollo sta nel fatto che una spia può ascoltare tutto cioè che transita nel canale, ma

per riuscire a calcolare i valori di a e b deve risolvere il problema del logaritmo discreto, computazionalmente troppo oneroso se i numeri in gioco sono molto grandi.

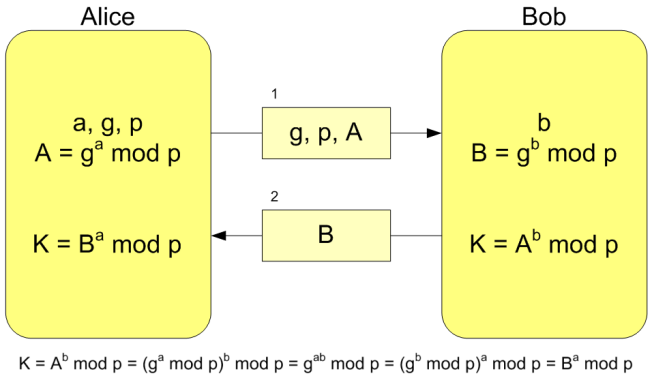

Figura 1.1: Scambio di chiavi Diffie-Hellman, schema di funzionamento.

TLS

Il TLS (Transport Layer Security) è un protocollo crittografico, utilizzato su reti TCP/IP per cifrare delle comunicazioni tra due nodi. Esso viene ampiamente usato per proteggere e-mail, VoIP, web browsing ecc, quindi si occupa di cifrare le connessioni a livello di applicazione del modello ISO/OSI. Al suo interno sono stati implementati vari algoritmi per scambiare le chiavi in maniera sicura e per l'autenticazione come RSA e Diffie-Hellman, AES per la cifratura simmetrica, MD5 e SHA per il controllo dell'integrità dei dati. Quando due sistemi vogliono comunicare tra loro attraverso TLS devono procedere in tre fasi:

- handshake e negoziazione tra le parti sugli algoritmi di cifratura da utilizzare (scambio della cipherlist o cipher suite);
- scambio delle chiavi e autenticazione;
- scambio di messaggi cifrati in maniera simmetrica e autenticati.

HTTPS

L'HTTPS (HyperText Transfer Protocol over Secure Socket Layer) è un protocollo per la comunicazione sicura, largamente utilizzato su internet. In pratica HTTPS consiste nella comunicazione tramite il protocollo HTTP all'interno di una connessione criptata da TLS. HTTPS garantisce l'autenticazione del sito web visitato, cifratura dei dati scambiati, protezione della privacy e integrità dei dati.

SHA

SHA (Secure Hash Algorithm) indica una famiglia di cinque algoritmi crittografici, utilizzati per produrre una sorta di impronta digitale del messaggio, chiamato hash. Se indichiamo con M un messaggio qualsiasi e con $\mathcal{H}$ la funzione di hash, allora $D = \mathcal{H}(M)$ sarà diversa per ogni $M' \neq M$. Generalmente algoritmi di questo tipo vengono utilizzati per verificare l'integrità dei messaggi. Inviando l'hash di un messaggio, insieme al messaggio stesso, il destinatario può verificarne l'integrità ricalcolando la funziona e confrontando il risultato con quello ricevuto dal mittente. Se il messaggio subisce modifiche prima di arrivare a destinazione, l'hash calcolato a destinazione sarà diverso da quello calcolato dalla sorgente.

2 La rete Tor

I messaggi trasmessi attraverso l'Onion Routing vengono crittografati a strati con metodi di crittografia simmetrica e inoltrati da dei router o relay, chiamati *Onion Router* (*OR*), in cui è in esecuzione un processo a livello utente, che non necessita di privilegi particolari. Ogni client che vuole stabilire una connessione sicura con un host, deve prima costruire un tunnel attraverso la rete Tor utilizzando un particolare protocollo. Viene detta crittografia stratificata perché man mano che il messaggio avanza nel tunnel, viene rimosso un livello crittografico, come spiegato nella Figura 2.1. Gli utenti che vogliono collegarsi alla rete devono utilizzare un software chiamato *Onion Proxy* (*OP*) che stabilisce i circuiti all'interno della rete e gestisce le connessioni. Ogni router possiede una chiave a lungo termine, utilizzata per firmare i certificati TLS e le proprie caratteristiche che vengono divulgate pubblicamente (come bitrate, indirizzo, tipo ecc.). Inoltre possiede una chiave pubblica a breve termine o onion key usata dai client per crittografare le richieste che, come già detto, viene cambiata periodicamente.

Le cell

La maggior parte dei dati scambiati tra due router o tra un router e un client avviene attraverso delle *cell* di dimensione fissa. Esse sono dei pacchetti a livello di applicazione di dimensione fissa di 512 byte, formate da un header e da un payload. L'header contiene informazioni come il circID che serve per identificare il circuito al quale la cell è destinata o anche il CMD che indica cosa fare con il payload. Tutte le comunicazioni router-router e router-client avvengono attraverso delle connessioni TLS (più circuiti possono essere multiplati attraverso una connessione TLS). Esistono vari comandi che vengono inseriti nel campo CMD di una cell, ad esempio `CREATE` o `CREATED` usati per costruire un nuovo circuito, oppure `RELAY` che indica che la cell deve essere solo inoltrata al prossimo nodo ecc. Le cell di tipo `RELAY` possiedono dei campi aggiuntivi all'interno dell'header come lo StreamID che identifica lo stream tra tutti quelli multiplati nel circuito oppure il Digest che serve per verificare l'integrità dei dati.

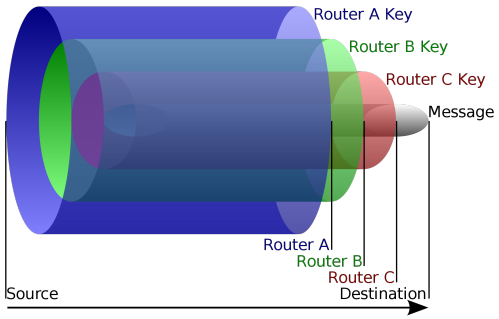

Figura 2.1: Livelli di crittografia dell'onion routing.

2.1 Scelta dei nodi e costruzione del circuito

L'Onion Proxy costruisce in maniera incrementale il circuito, scegliendo solitamente tre Onion Router tra tutti quelli disponibili. Per sapere quali relay sono disponibili in un dato momento, esistono dei *directory server* che contengono una lista di tutti gli *OR*, comprensiva di indirizzo IP, bitrate offerto, chiave pubblica e altre informazioni. Come

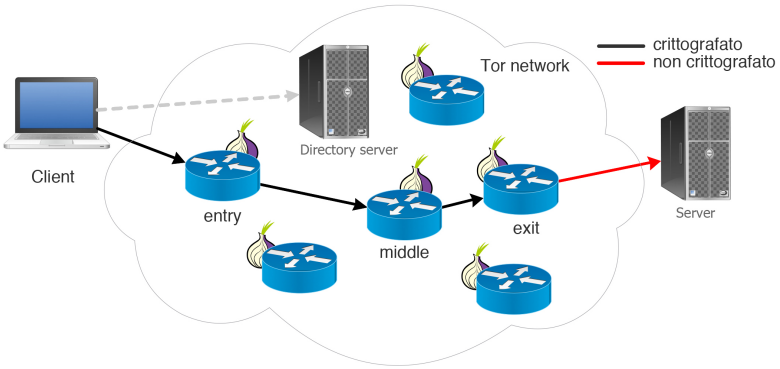

Figura 2.2: Circuito Tor.

mostrato in Figura 2.2 ogni circuito è costituito da un entry router, un middle e un exit. Per assicurare buone prestazioni essi vengono scelti tramite un algoritmo pesato di selezione che tiene conto del bitrate che ogni router annuncia. I router possono essere suddivisi in quattro insiemi diversi: pure entry (o entry guard) (G_0), pure exit (E_0), sia entry che exit (A) e né entry né exit (N_E). Indichiamo con i l' i-esimo router all'interno della rete con $i = 1, 2, ..., N$ dove N è il numero totale di onion router. Possiamo dare quindi le seguenti definizioni:

$$E_0 = \{i : OR_i \text{ è pure exit}\} \subset E \text{ con } E = \{i : OR_i \text{ è exit}\} \subset \{1,\ 2,...,N\} \tag{2.1.1}$$

$$G_0 = \{i : OR_i \text{ è pure entry}\} \subset G \text{ con } G = \{i : OR_i \text{ è entry}\} \subset \{1,\ 2,...,N\}. \tag{2.1.2}$$

Da cui derivano le relazioni:

$$E = A \cup E_0 \text{ e } G = A \cup G_0 \tag{2.1.3}$$

che indicano che gli entry router sono formati dall'unione dei pure entry e da quelli che possono operare sia da entry che da exit; la stessa cosa vale per gli exit. Se B_i rappresenta il bitrate dell' i-esimo router (uguale sia in ingresso che in uscita), il bitrate totale della rete sarà definita come:

$$B = \sum_{i=1}^{N} B_i \tag{2.1.4}$$

mentre il bitrate totale di uscita, puramente di uscita, di ingresso e puramente di ingresso sono rispettivamente:

$$B_E = \sum_{i \in E} B_i \ , \ B_{E_0} = \sum_{i \in E_0} B_i \ , \ B_G = \sum_{i \in G} B_i \ , \ B_{G_0} = \sum_{i \in G_0} B_i.$$

Per primo viene scelto il router di uscita e quindi bisogna definire un peso per il bitrate dei router $\in A$ (in quanto anche essi possono essere scelti come exit):

$$W_G = \begin{cases} 1 - \frac{B}{3B_G}, & altrimenti, \\ 0, & \text{se } B_G < B/3. \end{cases} \tag{2.1.5}$$

Allora la probabilità di scegliere l' i-esimo router, come router di uscita è definita come:

$$P_i = \begin{cases} \frac{B_i}{B_{E_0} + W_G B_A}, & i \in E_0 \\ \frac{W_G B_i}{B_{E_0} + W_G B_A}, & i \in A. \end{cases} \tag{2.1.6}$$

Questo significa che, se ad esempio, il bitrate in ingresso è scarso ($B_G < B/3$), i router che possono operare sia da entry che da exit, non verranno considerati nel momento in cui verrà scelto il nodo di uscita. Per la scelta del router d'ingresso vale lo stesso ragionamento:

$$W_E = \begin{cases} 1 - \frac{B}{3B_E}, & altrimenti, \\ 0, & \text{se } B_E < B/3 \end{cases} \tag{2.1.7}$$

e la probabilità di scegliere un router come entry diventa:

$$P_i = \begin{cases} \frac{B_i}{B_{G_0} + W_E B_A}, & i \in G_0 \\ \frac{W_E B_i}{B_{G_0} + W_E B_A}, & i \in A \end{cases} \tag{2.1.8}$$

con $B_A = \sum_{i \in A} B_i$. Dopo aver scelto i router di uscita e di ingresso, si procede con la scelta del middle. Anche in questo caso vanno valutati i pesi prima di calcolare la probabilità che venga scelto l'i-esimo router:

$$\bar{B}_i = \begin{cases} W_E B_i, & i \in E_0 \\ W_G B_i, & i \in G_0 \\ W_E W_G B_i, & i \in A \\ B_i, & i \in N_E \end{cases}, \; P_i = \frac{\bar{B}_i}{B_{E_0} W_E + B_{G_0} W_G + B_A W_E W_G + B_N} \tag{2.1.9}$$

Una volta scelti i nodi che andranno a costituire il circuito, il client (Alice nella Figura 2.3) invia una `CREATE` cell al router d'ingresso, contenente la prima metà dell'handshake (g^x) di Diffie-Hellman, crittografata con l'onion key. L'OR_1 deve quindi decifrare la richiesta e rispondere con una `CREATED` cell contenente (g^y) insieme all'hash della chiave negoziata $K = g^{(xy)}$. Da questo momento in poi Alice e l'entry router potranno comunicare utilizzando un algoritmo di crittografia simmetrica (tipicamente AES) con la chiave appena concordata. Per estendere il circuito, il client deve inviare una `RELAY_EXTEND` cell al router d'ingresso, comprensiva dell'indirizzo del prossimo relay e della prima metà della dell'handshake di Diffie-Hellman (g^{x_2}). L'entry copia il mezzo handshake in una `CREATE` cell e la inoltra al middle router. Quando l'OR_2 risponde con la `CREATED` cell, l'OR_1 copia il payload all'interno di una `RELAY_EXTENDED` cell e la invia ad Alice. Adesso anche Alice e il middle router condividono una chiave $K_2 = g^{(x_2 y_2)}$ da utilizzare per comunicare tra di loro. Per estendere ulteriormente il circuito, viene ripetuta la procedura appena descritta, comunicando il comando di estensione all'ultimo router. Normalmente, quando due router o un router e il client devono comunicare tra di loro, utilizzano le `RELAY` cell. Quando il client fa una richiesta, ad esempio ad un server web che si trova al di fuori della rete Tor, essa viene incapsulata in una `RELAY` cell, crittografata sequenzialmente con le chiavi di sessione e inviata all'entry. La cell deve percorrere il circuito dal primo all'ultimo router, ognuno dei quali si occupa di rimuovere un livello di crittografia. L'ultimo router, oltre a rimuovere l'ultimo livello, inoltra la richiesta al server e riceve la risposta. Una volta ricevuta la risposta, anch'essa viene incapsulata in una cell, crittografata con la chiave di sessione e rispedita al nodo precedente, proseguendo allo stesso modo attraverso gli altri nodi. Il nodo di uscita quindi esegue le operazioni per conto del client, aprendo una connessione in genere non crittografata con gli host esterni alla rete Tor. Grazie al meccanismo appena descritto, il client può rimanere in totale anonimato, poichè ogni router conosce solo i due nodi con i quali sta comunicando, ma non conosce gli altri componenti del circuito.

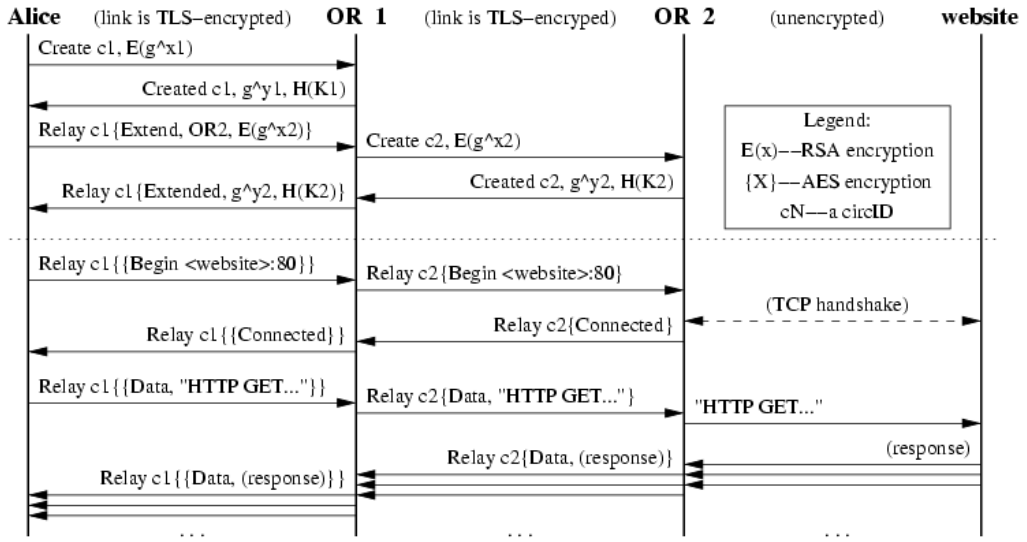

Figura 2.3: Procedura per la costruzione di un circuito e richiesta ad un server web.

2.2 Servizi Nascosti

Oltre a garantire l'anonimato ai client, Tor è in grado di fare la stessa cosa lato server. Questo significa che è possibile fornire un servizio (come ad esempio un sito web), facendo in modo che la localizzazione del server rimanga sconosciuta. Per accedere ad un servizio nascosto, un client deve obbligatoriamente utilizzare Tor, e deve riferirsi ad esso attraverso un pseudo dominio di primo livello ".onion". Quando un nuovo servizio viene attivato, esso deve comunicare la propria esistenza alla rete per poter essere raggiungibile dai client. Per prima cosa deve stabilire dei *punti di introduzione*, cioè scegliere in maniera casuale dei router e costruire dei circuiti che terminano in questi ultimi, comunicandogli di rimanere in attesa di richieste. Successivamente viene inviato ad un database, un *descrittore* (firmato con la chiave privata), contenente la chiave pubblica e un elenco dei punti di introduzione. Quando un client (Alice) tenta di contattare un servizio nascosto (Bob), vengono eseguite le seguenti operazioni:

1. Alice esegue una chiamata all'indirizzo xyz.onion, per ottenere il descrittore del servizio e conoscere così i suoi punti di introduzione (xyz è una sequenza di 16 caratteri derivante dalla chiave pubblica del servizio stesso);

2. Alice sceglie un OR come *punto di rendezvous* da utilizzare per connettersi al servizio;

3. Alice apre un circuito anonimo con uno dei punti di introduzione per comunicare al servizio il punto di rendezvous scelto e la prima metà dell'handshake di DH;

4. se Bob accetta la richiesta, si connette al punto di rendezvous e comunica ad Alice la seconda metà dell'handshake di DH e un hash della chiave concordata;

5. adesso il punto di rendezvous collega il circuito di Alice con quello di Bob, che potranno comunicare in maniera del tutto anonima.

3 Blanket blocking

Il *blanket blocking* è un attacco che permette di bloccare l'accesso all'intera rete Tor e viene solitamente attuato da enti governativi, ISP o amministratori di rete. Esso non intende colpire un router o un client specifico, ma attraverso l'utilizzo di firewall o filtri impedisce l'accesso a chiunque si voglia collegare alla rete Tor. Esistono principalmente due metodi per impedirne l'accesso:

- blocco delle connessioni verso le infrastrutture di Tor (*address-blocking*);
- utilizzo di tecniche DPI (*content-based blocking*).

Per attuare il primo metodo, un ISP (ad esempio) può semplicemente inserire in una blacklist tutte le coppie IP:porta delle directory authorities, gli IP distribuiti da esse o l'IP del server che ospita il sito `https://www.torproject.org/` (per impedire che il software si diffonda). Ogni qualvolta si tenterà di stabilire una connessione verso uno di questi indirizzi IP, verrà impedito ai pacchetti di raggiungere il server, oppure di ricevere le risposte. Con le tecniche DPI (Deep Packet Inspection) i pacchetti vengono analizzati da software specifici in grado di individuare il contenuto del pacchetto o l'applicazione che l'ha generato (in questo caso Tor) e scartarli per impedire che arrivino a destinazione. Nel caso di Tor, solitamente, viene individuato il protocollo nel momento in cui il client invia la cipher list al primo router, contenuta nel TLS hello[1]. Infatti la cipher list sembra essere unica ed utilizzata solamente da Tor.

3.1 Bridge

Il metodo più semplice per difendersi da questo tipo di attacco consiste nell'utilizzare i bridge. Essi sono dei router Tor, che non vengono inseriti nelle liste delle directory authorities. Dato che i loro indirizzi IP non sono pubblici, non vengono censurati e quindi è possibile utilizzarli come first-hop per collegarsi alla rete Tor. Per ovvi motivi vengono messi a disposizione dei metodi che permettono di ottenere solamente pochi bridge alla volta. Inviando un' email a `bridges@torproject.org` da un account Riseup, Gmail o Yahoo si ottiene una risposta con tre indirizzi IP di tre bridge diversi. Oppure vengono distribuiti attraverso server HTTPS come `https://bridges.torproject.org/`, social networks o canali privati.

Un metodo molto più sofisticato [4] per raccogliere gli indirizzi IP dei bridge, è quello di utilizzare un middle router controllato che raccolga informazioni sugli altri router della rete: inserendo all'interno della rete un router con un bitrate elevato si avrà un alta probabilità che esso venga scelto come middle nelle costruzioni dei path. Così facendo il router controllato può confrontare gli indirizzi IP dei nodi di ingresso con quelli delle directory authorities. Se l'indirizzo confrontato non è presente tra quelli pubblici allora il router d'ingresso è un bridge e quindi può essere comunicato attraverso un'email ad un indirizzo prestabilito. Quando gli onion router notano che il bitrate e l'uptime del router controllato stanno al di sopra di un valore medio, esso viene etichettato come entry router dai directory server. Per evitare che il nodo inserito all'interno della rete diventi un entry bisogna regolare il bitrate e l'uptime dinamicamente, mentre per evitare che diventi un exit è necessario indicarlo in fase di configurazione.
Analizziamo allora la **probabilità di cattura**, ovvero la probabilità che un circuito che sceglie come first-hop un bridge, attraversi il router controllato. Ipotizziamo di inserire k

[1] Il TLS hello è la prima fase dell'handshake TLS, in cui il client comunica al server (o al router in questo caso) gli algoritmi crittografici supportati, la versione TLS ecc.

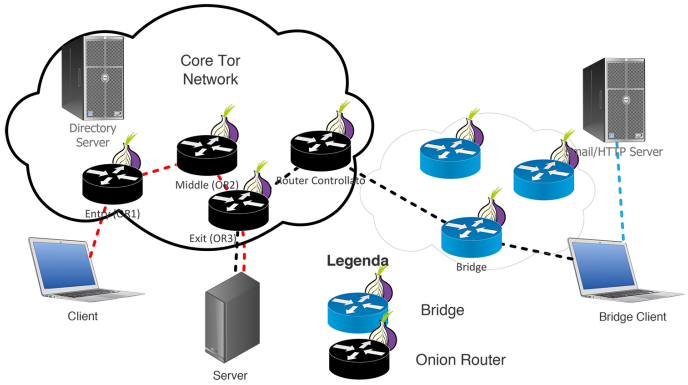

Figura 3.1: Scenario di rete con router controllato.

middle router nella rete e che il numero totale di OR fino a quel momento fosse N, allora i rispettivi bitrate li indicheremo con $\{B_1, B_2, ..., B_k, B_{k+1}, ..., B_{k+N}\}$. Tutti i router inseriti avranno lo stesso bitrate, $B_1 = B_2 = ... = B_k = b$. Definiamo B come la somma dei bitrate di tutti gli OR originariamente presenti nella rete: $B = \sum_{i=k+1}^{k+N} B_i$. Quindi dopo l'inserimento, il bitrate totale sarà $B + k \cdot b$. Da (2.1.7) e (2.1.5) possiamo scrivere:

$$W_E = \begin{cases} 1 - \frac{B+k\cdot b}{3\cdot(B_{E0}+B_A)}, & \text{se } W_E > 0, \\ 0, & \text{se } W_E \leq 0. \end{cases} \tag{3.1.1}$$

$$W_G = \begin{cases} 1 - \frac{B+k\cdot b}{3\cdot(B_{G0}+B_A)}, & \text{se } W_G > 0, \\ 0, & \text{se } W_G \leq 0. \end{cases} \tag{3.1.2}$$

Diamo allora la definizione di probabilità di cattura:

Teorema 1 (Probabilità di cattura)

$$P(k,b) = \frac{k \cdot b}{B_{E0} \cdot W_E + B_A \cdot W_E \cdot W_G + B_{G0} \cdot W_G + (B_N + k \cdot B)} \tag{3.1.3}$$

Da qui deriva immediatamente che più è grande il bitrate portato all'interno della rete, più probabilità si ha di essere scelti nei circuiti Tor, come enunciato dal seguente teorema.

Teorema 2 *La probabilità di cattura è determinata dall'ammontare del bitrate aggiunto dagli OR controllati. Sia $M = k \cdot b, M' = k' \cdot b'$ con $M \geq M'$,*

$$P(M) \geq P(M'). \tag{3.1.4}$$

L'uguaglianza si ha con $M' = M$.

Teorema 3 *Dopo la creazione di q circuiti, la probabilità che almeno uno dei bridge si sia collegato ad uno dei router controllati è:*

$$P(k,b,q) = 1 - (1 - P(k,b))^q, \;\; q = 1, 2, 3, ... \tag{3.1.5}$$

In conclusione, la probabilità di cattura aumenta con l'aumentare del bitrate totale dei router controllati e con il numero di circuiti che vengono creati. Come è stato evidenziato sperimentalmente [4], in pochi giorni con un solo router si è in grado di raccogliere un numero considerevole di bridge.

3.2 Pluggable transports

Il semplice uso dei bridge si oppone solo alla censura a livello IP e non va a contrastare, laddove vengono utilizzate, le tecniche DPI. Come abbiamo detto il Deep Packet Inspection va ad analizzare il contenuto dei pacchetti, ma è possibile camuffarli con dei tool detti *pluggable transports* in modo da aggirare i blocchi di tipo content-based. Possono essere scelte due strategie per camuffare i pacchetti:

- far sembrare il flusso di pacchetti qualcosa che non viene bloccato;
- far sembrare il flusso di pacchetti qualcosa di autorizzato.

Dei tool che realizzano la prima strategia sono: obfs2, obfs3, ScrambleSuit e obfs4 che cercano di camuffare i pacchetti facendo sembrare lo stream di byte un insieme uniforme di bit casuali, inoltre fanno in modo che non rimanga nessuna parte della comunicazione in chiaro, neanche nella fase di handshake TLS e nello scambio delle chiavi. Anche se è possibile decifrare lo stream, questo richiederebbe ulteriori risorse che non vengono impiegate nella maggior parte dei casi. Questo metodo richiede che il tool sia eseguito sia sul client che sul bridge a cui ci si vuole collegare. Nel secondo caso vengono usate tecniche steganografiche, che "trasformano" lo stream di dati di Tor, in uno stream simile a quello di altre applicazioni com ad esempio Skype.

3.3 Great Firewall of China (GFC)

Great Firewall of China è un termine utilizzato per far riferimento al *Golden Shield Project*, un progetto per il controllo e la censura di Internet sviluppato dal governo cinese. Esso merita un approfondimento a parte per quanto riguarda l'aspetto tecnico data la sua efficacia nel realizzare il blanket blocking. Ogni volta che un client all'interno del territorio cinese tenta di collegarsi ad un OR, un blocco DPI identifica il protocollo Tor durante lo scambio della cipher suite TLS e lo comunica a dei server. Questi server provano a collegarsi all'OR con lo stesso protocollo e, se ci riescono, lo inseriscono in una blacklist. Questo vale anche quando qualcuno ci si collega ad un bridge, rendendoli quindi inutilizzabili. I directory authorities vengono bloccati a livello IP, infatti non rispondono né a pacchetti TCP né ICMP. Attualmente il GFC sembra lasciar passare il TCP SYN, ma eliminare il SYN/ACK inviato dal router al client e la stessa cosa sembra succedere quando si prova a collegarsi ad un bridge bloccato. Comunque il blocco di router e bridge sembra riferirsi alle tuple IP:porta, dato che essi rispondono al ping verso altre porte. Probabilmente questa scelta è stata fatta per minimizzare gli effetti collaterali dovuti alla censura.

3.3.1 Contromisure

I risultati sperimentali [5] mostrano che una tupla IP:Porta rimane bloccata fintanto che è raggiungibile. Questo significa che periodicamente i server del GFC provano a stabilire delle connessioni con i router/bridge che avevano precedentemente inserito nella blacklist. Se vedono che non sono più raggiungibili rimuovono il blocco. Questo meccanismo può essere sfruttato per aggirare il firewall. Si può pensare di inserire un router all'interno della rete Tor posizionato al di fuori del territorio cinese e configurarlo in modo che accetti solo connessioni da degli indirizzi IP prestabiliti. Così facendo anche se i blocchi DPI rilevassero l'uso del protocollo Tor verso il router, i server del GFC non potrebbero provare a collegarsi dato che le loro richieste verrebbero scartate e quindi non bloccherebbero il router.

Il progetto Tor ha anche sviluppato un tool chiamato *meek* [6, 7] in grado di codificare uno stream di dati come una sequenza HTTPS di richieste e risposte che vengono inoltrate

da un web server; questa tecnica è detta *domain fronting*. Il server quindi ha la funzione di

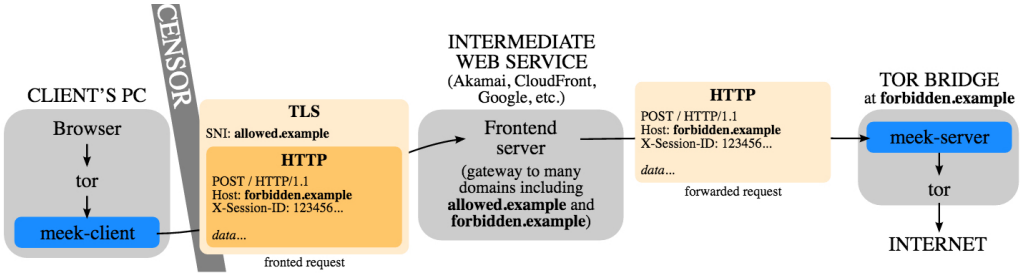

Figura 3.2: Schema di funzionamento del tool meek.

intermediario e permette ad un client di collegarsi ad un Tor bridge, facendo sembrare che le richieste vadano verso un IP autorizzato. Meek si compone di due parti: meek-client e meek-server. La parte client è un processo separato rispetto al browser Tor che riceve i dati da trasmettere da esso, li "impacchetta" dentro ad una richiesta POST e li invia al bridge attraverso il web server. Il dominio autorizzato viene messo "all'esterno" della richiesta: nella query DNS e nell'estensione SNI TLS [2], mentre il dominio vietato "all'interno": nell' host header della richiesta HTTP. Sul bridge è in esecuzione la parte server di meek, che decodifica le richieste HTTP in ingresso e le invia nella rete Tor. Dopo aver ricevuto una richiesta, meek-server controlla se il bridge ha dei dati da inviare come risposta al client, e li invia nella risposta HTTP; quando meek-client riceve la risposta, ne passa il corpo al processo Tor. Il motivo per cui questi web server non vengono bloccati è che fanno parte di qualche CDN, chi censura quindi, incapace di distinguere tra traffico fronted e non fronted, è costretto a scegliere: permettere il passaggio di traffico che aggira i firewall o bloccare completamente interi domini, con pesanti effetti collaterali.

[2]L'estensione SNI (Server Name Indication) è un'estensione del protocollo TLS con cui un client indica a quale hostname sta cercando di connettersi all'inizio della fase di handshake.

4 Targeted Attack

I *targeted attack*, al contrario del blanket blocking, non riescono a bloccare completamente l'accesso alla rete, ma possono comunque causare un calo di prestazioni, di affidabilità e di sicurezza rendendo inutilizzabili i nodi più importanti della rete. Nelle sezioni successive verranno descritti vari tipi di attacchi, le loro conseguenze e, per ognuno, discusse delle specifiche contromisure.

4.1 CellFlood Attack

Il CellFlood [8] è un attacco DoS selettivo, che permette di sovraccaricare la CPU di un Onion Router in maniera molto rapida, richiedendo un basso costo in termini di risorse alla macchina dell'aggressore. Esso sfrutta una vulnerabilità nel protocollo di Tor, che si manifesta nella fase di costruzione di un circuito. Gli autori dello studio affermano che questo attacco è in grado di *dimezzare* la capacità di processing del 62% degli OR più usati di tutta la rete; inoltre richiede solamente un bitrate compreso tra 2.6 e 9.76 Mb/s per router che è molto inferiore rispetto a quello richiesto da un classico attacco DoS.

Ricordiamo che, ogni volta che si aggiunge un OR al circuito, il client invia una `RELAY_EXTEND` cell all'ultimo router OR_i contenente un'*onionskin* indicata con $E(g^x)$, dove $E(\cdot)$ indica la crittografia con l'*onion key* pubblica RSA a 1024-bit, precedentemente scaricata da un'authority Tor. Una volta ricevuta la cell, l'OR_i estrae l'onionskin e la invia al router da aggiungere OR_{i+1} nel payload di una `CREATE` cell. Una volta che l'OR_{i+1} decifra l'onionskin, risponde all'OR_i con una `CREATED` cell, che viene incapsulata e inoltrata al client in una `RELAY_EXTENDED` cell.

L'operazione di processing dell'onionskin, è molto più costosa in termini di risorse rispetto alla sua creazione da parte del client, infatti decifrare con una chiave privata a 1024-bit occupa circa 20 volte il tempo necessario a cifrare con la corrispondente chiave pubblica a 1024-bit. Per questo motivo si spende $\sim$ 4 volte più tempo a processare una `CREATE` cell piuttosto che generarla. Tutto questo può essere sfruttato da un client aggressore che può consumare tutte le risorse di un Onion Router sommergendolo di `CREATE` cell. Inoltre non è necessario che ogni volta il client calcoli le nuove onionskin, ma può utilizzare sempre la stessa dato che l'OR sarà costretto comunque a processarle una ad una. Nel software Tor in esecuzione negli OR, il main thread si occupa del più importante compito di inoltrare le `RELAY_DATA` cell, mentre uno o più thread, chiamati CPU Workers, si occupano di processare le onionskin. Questa scelta di progettazione permette all'Onion Router di continuare ad inoltrare le cell anche se nello stesso momento viene inondato di `CREATE` cell. Se però il rate di arrivo diventa più alto di quanto la CPU è in grado di processare, allora esse vengono scartate e viene risposto con una `DESTROY` cell. Questo causa un interruzione del servizio per tutti i client legittimi che vogliono stabilire una connessione con l'OR. Quindi se il CellFlood venisse utilizzato in maniera strategica, contro un insieme di router importanti, la parte rimanente della rete verrebbe sovraccaricata, oppure i tunnel costruiti da quel momento in poi potrebbero passare per degli OR controllati, con una conseguente diminuzione di affidabilità, sicurezza e QoS. Un aggressore interessato ad attaccare un router, oppure un insieme di router, avrebbe a disposizione una tecnica più potente e meno costosa rispetto ad un classico attacco DoS a livello di rete.

4.1.1 Difesa attraverso l'uso dei Client Puzzles

Come contromisura al CellFlood Attack, sempre gli autori di [8] propongono una soluzione basata sui *client puzzles*. Per fare in modo che il carico sulla CPU non vada fuori controllo,

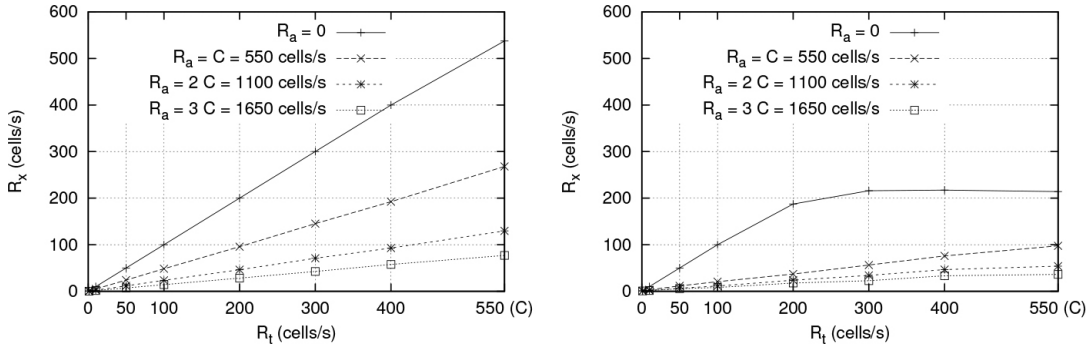

Figura 4.1: Effetto di un CellFlood su un OR, a sinistra senza traffico dati, mentre a destra con traffico dati. R_t indica il rate di cell benigne, R_a dell'aggressore, R_x quelle effettivamente processate.

è stato pensato di far processare le onionskin al router, solo dopo che il client ha risolto un problema (tipicamente crittografico) che richiede una certa quantità di risorse computazionali. Costringere il client a risolvere ogni volta un problema di questo tipo, riduce in maniera importante l'efficacia dell'attacco. I client puzzles sono un'ottima soluzione per difendere gli OR per vari motivi:

- ogni router può difendersi indipendentemente, senza cooperare con gli altri OR, con il rischio che un altro qualsiasi possa essere controllato;
- l'efficacia dei client puzzles non dipende da come l'attacco è stato organizzato: può arrivare direttamente da un altro router o da un client, oppure indirettamente attraverso un altro router, incapsulando le onionskin in una `RELAY_EXTEND` cell invece che in una `CREATE` cell;
- le connessioni attive sul router sotto attacco non vengono chiuse;
- possiedono un livello di difesa modulare, nel senso che la loro complessità può essere modificata in base all'entità dell'attacco, rimanendo un'ottima soluzione anche in caso di un attacco DDoS.

Funzionamento

I client puzzles sono basati sugli HMAC (keyed-hash message authentication code) con SHA-256. Per i nostri scopi ci basta sapere che la funzione HMAC, è univoca per ogni coppia s ed m, dove s è la chiave, m il messaggio e X il risultato della funzione: X=HMAC(s, m). Per costruire un puzzle l'OR genera una chiave casuale s a 64 bit e calcola il valore X=HMAC$(s, P|H)$, dove $P|H$ è il messaggio risultante dalla concatenazione dell' onionskin P, contenuta nel payload della `CREATE` cell e l'hash[1] H della chiave pubblica[2] a lungo termine del router. Il puzzle è composto dalla tripletta (s', k, X) dove s' è la chiave s con gli ultimi k bit impostati a 0. Esso quindi non include il messaggio $P|H$, dato che il client è a conoscenza sia di P che di H. Una volta che il puzzle viene inviato al client, esso deve risolverlo cercando di indovinare il valore degli ultimi k bit di s, che deve per forza essere fatto per tentativi. Per ogni tentativo deve provare una combinazione s'', calcolare la funzione X'=HMAC$(s'', P|H)$ e verificare se $X' = X$. Quando l'uguaglianza è verificata,

[1]Una funzione di hash, è un algoritmo crittografico che trasforma una stringa binaria di lunghezza variabile (messaggio), in una stringa di dimensione fissa; è detta anche impronta digitale del messaggio poiché a messaggi diversi corrispondono hash diversi.

[2]La chiave pubblica a lungo termine del router serve per identificarlo univocamente nella rete e viene stabilita al momento della configurazione.

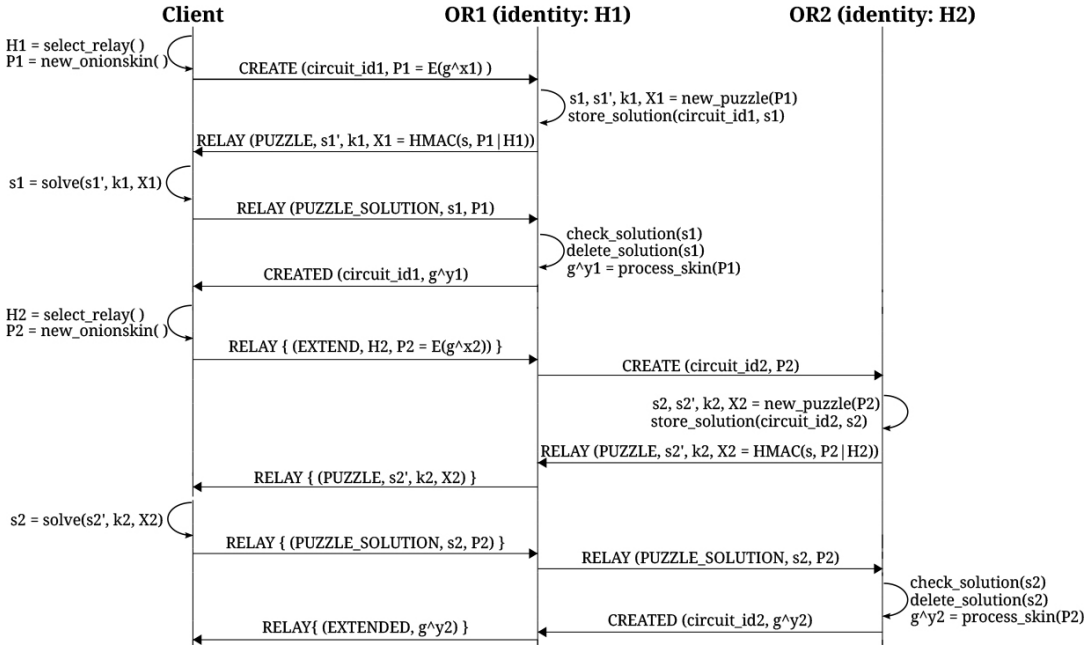

Figura 4.2: Funzionamento dei client puzzles.

allora ha indovinato il valore di s. Un puzzle di valore k richiede in media 2^{k-1} tentativi, e il tempo di calcolo cresce esponenzialmente con k. Questo permette di poter decidere se far lavorare il client pochi millisecondi fino ad arrivare a qualche ora.

Per scambiare i puzzles e le rispettive soluzioni, il client e il router inseriscono nel payload di una `RELAY` cell i comandi `PUZZLE` e `PUZZLE_SOLUTION`. Come descritto nelle specifiche di Tor, esse vengono crittografate con le chiavi di sessione precedentemente negoziate, quindi l'unico router che può intercettare sia il puzzle che la sua soluzione è l'ultimo tra quelli che fanno parte del circuito già creato; nel caso della figura 4.2 ad esempio, questo è l'$OR1$, nel momento in cui si vuole estendere il circuito fino a l'$OR2$. Come si può vedere dalla Figura 4.2 la sequenza delle operazioni necessarie ad aggiungere un nuovo router al circuito sono le seguenti:

1. Il client invia una `CREATE` o una `RELAY_EXTEND` cell per selezionare il nuovo router;
2. il router crea il puzzle, memorizza la soluzione e risponde con una `RELAY` cell inserendolo all'interno del payload;
3. il client risolve il puzzle e sempre tramite una `RELAY` cell comunica la soluzione attraverso il comando `PUZZLE_SOLUTION`;
4. il router controlla se la soluzione s è corretta e in caso affermativo comunica tramite una `CREATED` cell che la connessione è stata stabilita.

Memorizzare la soluzione di un puzzle, richiede 18 bytes di memoria (8 per la soluzione del puzzle, 8 per l'ID della connessione e 2 per l'ID del circuito). Quindi un potenziale aggressore potrebbe sfruttare questa piccola occupazione di memoria per condurre un attacco che tenta di consumare la RAM libera del router facendo molte richieste ad esso e lasciando i puzzle irrisolti. Per fare in modo che questo non accada, viene assegnata una scadenza Δ_p al puzzle, entro il quale deve essere risolto, altrimenti la soluzione viene eliminata dalla memoria. Scegliere un Δ_p adeguato non è difficile, in quanto anche scegliendolo pari a 2 minuti un aggressore che ha a disposizione un bitrate di 189 Mbit/s riesce a consumare solo 100 MB di memoria. Inoltre è stato verificato che la rete Tor supporta uno stream di `CREATE` cell di pochi Mbit/s. Un Δ_p pari a 2 minuti risulta essere adeguato anche per i

client più lenti, che riusciranno quindi a risolvere il puzzle in tempo. Inoltre il valore di Δ_p può essere modificato a seconda della situazione. Una strategia alternativa potrebbe essere quella di scegliere una sola chiave s che dopo un certo periodo di tempo viene sostituita, in modo da evitare che un aggressore possa utilizzarla più volte. Questo permette di memorizzare solo una soluzione, ma scegliere il periodo di scadenza è un'operazione difficile che richiede una stima accurata del bitrate a disposizione dell'aggressore e della velocità con cui i client benigni riescono a risolvere il puzzle. Un valore troppo grande rischia di essere pericoloso, mentre troppo piccolo svantaggia i client onesti.

Scelta della difficoltà del puzzle

Si possono seguire due approcci differenti nell'implementazione dei client puzzle: inviarli ai client, in qualunque situazione con un livello di difficoltà fissata oppure inviarli solo se il router diventa obiettivo di un attacco. Il primo approccio è più facile da implementare ma richiede risorse inutili ai client benigni, mentre il secondo non comporta l'utilizzo di risorse inutili ma per essere efficace bisogna essere in grado di capire se il router sta subendo un attacco DoS. Per capire se l'OR è sotto attacco, si può pensare di contare ogni Δ_x secondi il numero P_x di `CREATE` cell che è stato in grado di processare e il numero D_x che invece sono state scartate dato che i CPU workers erano occupati. È possibile quindi calcolare il valore $\overline{\mu} = \frac{D_x}{P_x}$ e controllare ogni Δ_x secondi se la percentuale di richieste scartate sta sopra o sotto una certa soglia β. La prima volta che $\overline{\mu}$ supera il valore di soglia β, il router inizia ad inviare i puzzle a tutti i client che desiderano aggiungerlo al proprio circuito, con una difficoltà iniziale di parametro k (i.e. $k = 16$) per Δ_x secondi. Alla fine dell'intervallo l'OR aumenta di 1 il valore di k se $\overline{\mu} > \beta$, mentre lo diminuisce di 1 se $\overline{\mu} < \beta$; questo permette di variare dinamicamente la difficoltà del puzzle. Per evitare invece di assegnare un carico di lavoro troppo elevato ai client onesti, si può fissare un limite per k (ad esempio 20). Il parametro β dovrebbe essere fissato ad un valore basso, molto vicino allo 0. La scelta va fatta tenendo conto di quante cell i CPU workers potrebbero scartare durante una situazione di normalità. Infine il valore di Δ_x non è critico, basta che permetta di scegliere la difficoltà dell puzzle correttamente.

4.2 Sniper Attack

Lo *Sniper Attack* è potenzialmente uno degli attacchi DoS più distruttivi ed efficaci, infatti *permette anonimamente di disabilitare qualsiasi nodo Tor utilizzando pochissime risorse.* L'attacco può essere anche usato per identificare un servizio nascosto, mediante la disabilitazione selettiva dei relay. Esso si basa sul livello di congestione e sul meccanismo di controllo del traffico di Tor per causare un aumento incontrollato del contenuto del buffer dell'obiettivo. L'idea di base è quella di costruire un normale circuito Tor, utilizzando il nodo da colpire come entry e iniziare a scaricare un file di grosse dimensioni. Contemporaneamente si smette di leggere dal buffer in input e si invia con una certa frequenza il comando `SENDME` al nodo di uscita, per costringerlo ad immettere dati nel circuito. La procedura può essere ripetuta per vari circuiti, utilizzando sempre l'obiettivo come nodo d'ingresso. Quando la memoria del nodo sarà esaurita, il processo Tor verrà terminato dal sistema operativo del nodo, causandone l'inoperabilità. Secondo gli studi di [9], un attacco di questo tipo sarebbe in grado, in soli 29 minuti, di disabilitare i 20 più importanti nodi di uscita di tutta la rete; tutto questo inoltre con un costo bassissimo per chi conduce l'attacco. Infatti è possibile costringere l'obiettivo a consumare fino a 2187 KiB/s di memoria utilizzando un bitrate di solo 92 KiB/s in upstream e 39 KiB/s in downstream.

Per capire a fondo l'attacco però, è necessario prima conoscere come Tor gestisce il traffico nel circuito. Tor implementa un meccanismo a finestra variabile per controllare la

quantità di dati che viene immessa nella rete. Ogni nodo estremo del circuito (i.e. client ed exit) tiene traccia con due contatori di due finestre. Il *package* window, inizializzato al valore 1000, controlla quante data cell un *OR* è autorizzato a "impacchettare" e inserire all'interno del circuito, mentre il *delivery* window, inizializzato al valore 100, quante data cell è disposto a consegnare al di fuori di esso. Il primo contatore viene decrementato di uno quando una data cell viene *inserita* all'interno del circuito, mentre il secondo per ogni data cell che viene *rimossa*. Per ogni circuito possono esistere più stream: in questo caso si dice che gli stream sono multiplati; i circuiti invece sono multiplati sulle connessioni. Analoghi contatori esistono per ogni stream, quindi avremo un *package* window, questo inizializzato a 500, e un *delivery* window, inizializzato a 50. Il packaging edge (PE) di un circuito smette di introdurre data cell da uno qualsiasi degli stream multiplati il cui package window raggiunge il valore zero, oppure interrompe tutti i flussi di data cell di tutti gli stream quando il package window del circuito va a zero. Il delivery edge (DE) manda un comando di feedback chiamato `SENDME` al PE quando la delivery window di un circuito o quella di uno stream raggiungono il valore zero. Questo comando riporta il valore del delivery window del DE e del package window del PE al loro rispettivo valore iniziale[3]. Quindi per un dato stream non ci saranno più di 500 data cell all'interno della rete, e non più di 1000 per un singolo circuito.

Attacco basilare

Un DE che smette di leggere dal buffer di una connessione causa l'accodamento di un intera package window proveniente da un PE (1000 cell), assumendo che per ogni circuito attivo ci siano almeno due stream multiplati e che la quantità di dati che viene trasferita sia sufficiente a portare il package window del PE a zero.

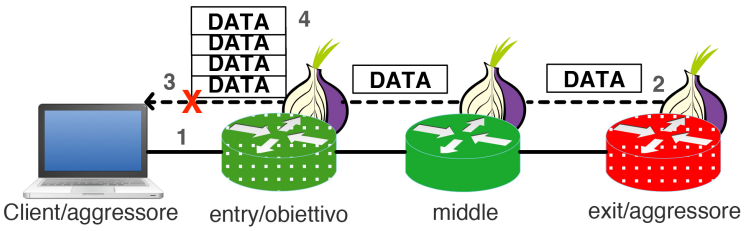

Figura 4.3: Sniper Attack basilare.

L'attacco si compone quindi delle seguenti fasi:

1. Il client costruisce un circuito selezionando l'obiettivo come entry e un relay complice come exit;
2. Il nodo di uscita genera pacchetti e li invia al client attraverso il circuito, ignorando il limite del package window;
3. Il client smette di leggere dalla connessione TCP stabilita con l'obiettivo;
4. Il buffer dell'obiettivo inizia a riempirsi fino a quando il sistema operativo non termina il processo Tor.

[3]In pratica il package window di circuito e di stream sono inizializzati rispettivamente a 1000 e a 500, mentre i delivery window a 100 e 50. Quando il PE invia una cell verso il DE, vengono decrementate di 1 sia i delivery window del DE che i package window del PE (sia dello stream corrispondente che del circuito). Normalmente, proseguendo in questo modo, il delivery window di uno degli stream o quello del circuito raggiungerà il valore zero. Questo significa che il package window dello stream avrà valore 450 oppure che il delivery window del circuito avrà valore 900. Il segnale di `SENDME` riporterà i valori dei delivery window a 100 e a 50, mentre quelli dei package window a 1000 e a 500.

Una piccola variante dell'attacco appena descritto consiste nell'utilizzare un file server come complice. In questo caso è il client a generare una grande quantità di dati da inviare al server che contemporaneamente smetterà di leggere dal buffer TCP, causando il riempimento della memoria e conseguentemente la terminazione del processo dell'exit router. Con questa variante dell'attacco è quindi possibile colpire un exit router.

Da notare il fatto che con *entrambe* le varianti si può colpire anche un *middle* router: nella prima versione è sufficiente avere come complice un entry relay che smette di leggere dal buffer della connessione con il middle router; nella seconda basta un exit relay che anch'esso smette di leggere dal buffer TCP.

4.2.1 Attacco avanzato

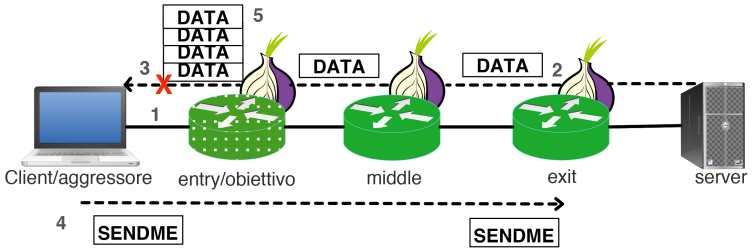

Figura 4.4: Sniper Attack avanzato.

L'attacco avanzato non necessita di un router/server complice e si basa su:

- interruzione della lettura dalla connessione TCP;
- utilizzo del comando `SENDME`.

Per prima cosa il client-aggressore (DE) deve stabilire un circuito, selezionando come entry router l'obiettivo da colpire. Successivamente deve iniziare a scaricare un file di grandi dimensioni. Una volta che il client smette di leggere dalla connessione stabilita con l'obiettivo, inizia ad inviare il comando `SENDME` al nodo di uscita (PE), in modo da evitare che la package window raggiunga il valore zero, smettendo così di immettere pacchetti nel circuito. Tor però possiede un meccanismo di protezione contro i client che provano ad inviare continuamente il comando `SENDME` per ottenere pacchetti con una maggiore frequenza: quando un exit riceve un `SENDME` che tenta di far alzare il valore della package window sopra a quello iniziale (1000), si accorge della violazione, chiude il circuito e invia un `DESTROY` al nodo precedente che lo inoltra all'entry e poi al client. Quando il client riceve il comando `DESTROY`, chiude anch'esso il circuito e lo inoltra all'entry e al middle che fanno la stessa cosa. Se questo accade, l'obiettivo, chiudendo il circuito, svuota il buffer e quindi l'attacco non va a buon fine. Per condurlo con successo bisognerebbe evitare che il contatore package window del nodo di uscita superasse la soglia iniziale, e che il client non inviasse il comando di eliminazione del circuito. Solo la seconda condizione si verifica dato che il client non leggerà dalla connessione con l'entry relay e quindi non riceverà, né inoltrerà il comando `DESTROY`. Bisogna quindi inviare il comando `SENDME` con una frequenza né troppo alta, che causa l'eccedere del package window, né troppo bassa che rende inefficace l'attacco. Per calcolare ogni quanto inviare il `SENDME` al nodo di uscita è sufficiente stimare il throughput S della connessione scaricando un file di una certa dimensione. Se vengono trasferiti σ KiB in Δ secondi allora $S = \frac{\sigma}{\Delta}$ KiB/s o anche $S = \frac{2\sigma}{\Delta}$ cells/s, dato che una cell Tor è di 512 byte. Il comando va inviato ogni 50 cell ricevute a livello di stream e ogni 100 ricevute a livello di circuito, ma senza leggere dalla connessione è impossibile contarle. Con le stime del throughput è possibile però sapere ogni quanti secondi inviare il comando: $T_{ss} = \frac{25\Delta}{\sigma}$ e $T_{cs} = \frac{50\Delta}{\sigma}$.

Dato che il throughput della connessione è un parametro che può variare facilmente, è possibile che il client invii troppo frequentemente il `SENDME`, causando l'avvio del meccanismo di protezione. Questo, anche se non provoca lo svuotamento del buffer dell'entry/obiettivo, non permette più di utilizzare quel circuito per condurre l'attacco. Per risolvere questo problema e per diminuire il tempo necessario a riempire il buffer dell'entry è possibile parallelizzare l'azione offensiva. Per fare ciò si possono utilizzare *team* di circuiti, ad ognuno dei quali è assegnato un path Tor. Per ogni team uno dei circuiti si occuperà di stimare il throughput, mentre gli altri di inviare il comando `SENDME`, utilizzando due processi Tor separati per team che svolgono le due funzioni. Infine, dato che non si hanno informazioni sullo stato dei circuiti creati, è possibile crearne di nuovi per assicurarsi che il consumo di memoria dell'obiettivo continui regolarmente.

Manca un altro aspetto da trattare (forse il più importante di tutti): mantenere l'anonimato. Condurre l'attacco senza essere scoperti è una delle cose più importanti e Tor può essere utile in questo. Un modo, detto *diretto* è quello di usare un exit node come aggressore, ovvero essere in questa situazione: $\mathcal{E}_A\mathcal{C}_A \leftrightarrow \mathcal{G}_O \leftrightarrow \mathcal{M} \leftrightarrow \mathcal{E} \leftrightarrow \mathcal{S}$ dove $\mathcal{E}$ indica un nodo di uscita, $\mathcal{G}_O$ un nodo d'ingresso (obiettivo), $\mathcal{M}$ un middle, e $\mathcal{S}$ un server. Un obiettivo non è in grado di capire se il nodo di uscita è il responsabile dell'attacco o se sta solo operando come servitore. Lo svantaggio è che l'exit node, per essere tale, deve mettere a disposizione una certa quantità di risorse nella rete e quindi alla fine l'attacco ne richiederebbe più di quante effettivamente necessarie. Inoltre, dato che l'obiettivo sarebbe direttamente connesso con l'aggressore, conoscerebbe il suo indirizzo IP, che potrebbe essere utilizzato per rintracciarlo e localizzarlo. Alternativamente si può pensare di utilizzare un intero circuito per nascondere l'IP. Questa tipologia di attacco viene detta *anonima*. Ci si troverebbe quindi in questa situazione: $\mathcal{C}_A^2\mathcal{C}_A^1 \leftrightarrow \mathcal{G}^1 \leftrightarrow \mathcal{M}^1 \leftrightarrow \mathcal{E}^1 \leftrightarrow \mathcal{G}_O^2 \leftrightarrow \mathcal{M}^2 \leftrightarrow \mathcal{E}^2 \leftrightarrow \mathcal{S}$ dove "1" indica il circuito di ingresso nella rete e "2" il circuito vero e proprio dell'attacco. $\mathcal{C}_A^1$ smette di leggere dalla connessione, mentre $\mathcal{C}_A^2$ invia il comando `SENDME`. In una configurazione di questo tipo potrebbe essere difficile stimare il throughput della connessione, inoltre due circuiti creati in sequenza potrebbero far nascere dei sospetti.

Dato che i top 100 relay, secondo il bitrate, vengono selezionati con una probabilità del 40%, un attacco che colpisce anche un piccolo numero di questi ultimi ha un impatto significativo sulle prestazioni della rete. Sono state fatte delle simulazioni [9] su uno scenario reale e sono stati misurati i tempi impiegati a disabilitare alcune categorie significative di nodi/infrastrutture Tor. È stato valutato sia l'effetto di un attacco diretto, sia anonimo, ciascuno su macchine con 1GiB e 8 GiB di RAM. L'analisi mostra che con l'attacco diretto è possibile disabilitare in solo un minuto il top `FAST`[4] Guard e il top `FAST` Exit con 1GiB di RAM, mentre con 8 GiB di RAM i tempi diventano rispettivamente di 18 e 8 minuti. Inoltre in soli 29 minuti è possibile escludere i top 20 Exit, causando problemi al 35% dei path costruiti in tutta la rete.

4.2.2 Difesa contro lo Sniper Attack

Lo Sniper Attack sfrutta principalmente due problemi intrinseci di Tor: mancanza di un controllo di flusso e assenza di bound e check sulle code attive. Questi problemi possono essere risolti con delle tecniche che verranno discusse in seguito.

`SENDME` autenticati

Nelle sezioni precedenti è stato mostrato come i packaging edge non sono in grado di verificare che i delivery edge abbiano ricevuto le cell. Ogni packaged cell contiene un hash del suo contenuto così da permettere al client di rilevare degli errori sui bit. Si

[4] `FAST` è un flag assegnato dalle directory authorities ad un router che indica che esso è adatto a costruire circuiti con un bitrate elevato. [3]

Tabella 4.1: Simulazione Sniper Attack

		Tempo in H:M per consumare tutta la RAM			
		Diretto		Anonimo	
	Sel %	1 GiB	8 GiB	1 GiB	8 GiB
Top `FAST` Guard	1.7	0:01	0:18	0:02	0:14
Median `FAST` Guard	0.025	0:23	3:07	0:23	3:07
Bottom `FAST` Guard	1.9e-4	1:45	14:03	1:45	13:58
Top `FAST` Exit	3.2	0:01	0:08	0:01	0:12
Median `FAST` Exit	0.01	1:45	14:03	1:22	10:53
Bottom `FAST` Exit	6e-5	1:45	14:03	1:48	14:20
Top 5 Guards	6.5	0:08	1:03	0:12	1:37
Top 20 Guards	19	0:45	5:58	1:07	8:56
Top 5 Exits	13	0:05	0:37	0:07	0:57
Top 20 Exits	35	0:29	3:50	0:44	5:52
All Dir Auths	N/A	17:34	140:32	17:44	141:49

potrebbe fare in modo che un package edge richieda gli hash di ogni packaged cell nel comando `SENDME`, anche se un malintenzionato potrebbe scaricare un file di cui aveva precedentemente calcolato gli hash durante l'attacco così da aggirare il controllo. Per fare in modo che questo non accada il PE potrebbe inserire un byte casuale ogni 100 cell il cui hash dovrebbe essere calcolato e comunicato dal DE. La probabilità di indovinare il byte e quindi l'hash sarebbe di $1/256 \approx 0.39$ %. Questo metodo impedisce di avere più di 1000 cell contemporaneamente nel circuito e fornisce difesa contro l'attacco avanzato, anche se non evita totalmente i suoi effetti dato che ogni circuito con 1000 cell consuma 500 KiB. Inoltre non difende dall'attacco basilare dove sia il client che l'exit sono sotto il controllo del malintenzionato.

Bound sulla dimensione delle code

Prima di continuare è necessario specificare che Tor, per come è stato progettato, crea una coda per ogni circuito, gestita con una pura politica FIFO. Un unico servitore a turno sceglie una coda e inoltra delle cell.
Il secondo problema critico è quello della dimensione che le code del processo Tor possono assumere. Esse possono crescere indefinitamente senza nessun controllo. Il metodo più semplice per risolvere il problema è quello di impostare un limite sulla dimensione della coda; quando viene superato, il router riconosce una violazione del protocollo e chiude il circuito, impedendo che l'aggressore consumi tutta la memoria. Bisogna però trovare un limite adeguato: si potrebbe utilizzare un valore intorno al 1000 con delle tolleranze, dato che il numero di cell in transito non possono essere più di 1000. Comunque il bound sulla dimensione delle code, come i `SENDME` autenticati non difendono dagli attacchi paralleli, che quindi sarebbero ancora potenzialmente in grado di mandare in crash il processo Tor di un *OR*.

Eliminazione adattiva circuito

Per superare le limitazioni dei metodi precedenti, è stato sviluppato un altro meccanismo molto più sicuro. L'approccio utilizzato è quello di continuare a chiudere circuiti attivi fintanto che il *totale* della memoria utilizzata rimane sopra una certa soglia; il nuovo problema diventa quindi decidere quale circuito chiudere. Chiudendo il circuito con la coda più lunga un aggressore potrebbe creare molti circuiti con code brevi, causando l'eliminazione di quelli che non fanno parte dell'attacco ma di traffico ordinario. Gli autori

[9] quindi propongono di eliminare il circuito con la cell più vecchia in testa alla propria coda; è necessario quindi che ogni cell sia etichettata con un timestamp. Dato che ogni coda è gestita con una pura politica FIFO, la cell in testa sarà la più vecchia di quel circuito. Per questo un eventuale aggressore sarebbe costretto a leggere continuamente dai propri circuiti per mantenere "giovani" le cell, altrimenti dopo una certa quantità di tempo verrebbero chiusi.
Assumiamo che il bitrate B di un OR sia uguale sia in ingresso che in uscita e che n sia il numero di circuiti attivi sul router in un certo momento. Sarebbe ragionevole pensare che il throughput venga diviso equamente tra tutti i circuiti, ma questo non succede in pratica. Si può assumere quindi che ogni circuito abbia a disposizione un throughput di almeno:

$$r \geq \alpha \frac{B}{n} \tag{4.2.1}$$

con $0 < \alpha \leq 1$. Ipotizziamo inoltre che esista anche un upper bound sulla coda di dimensione Q, diciamo circa di 1000 cell. Osserviamo che se la lunghezza della coda di un circuito benigno non supera il limite Q e il suo rate medio è almeno r, allora il massimo tempo che una cell spende nella coda è di:

$$d_{max} = \frac{Q}{r} = \frac{Qn}{\alpha B}. \tag{4.2.2}$$

Quindi se con t_{now} ci riferiamo all'attuale punto nel tempo, allora le cell in testa alle code dei circuiti benigni devono avere un timestamp più grande di $t_{now} - d_{max}$. Un eventuale aggressore dovrebbe fare in modo che le cell non spendano più di d_{max} in coda; dovrebbero allora arrivare in un punto del tempo maggiore di $t_{now} - d_{max}$. Solo in questo modo un circuito benigno potrebbe essere chiuso al posto di quello dell'aggressore. Conseguentemente le cell del circuito di attacco devono arrivare ad intervalli di tempo d con $0 < d \leq d_{max}$. Sia M la quantità di memoria libera del router prima dell'inizio di un eventuale attacco. L'aggressore dovrebbe riuscire a consumare una quantità M di memoria per causare l'eliminazione di circuiti benigni. Dato che il tempo che intercorre tra gli arrivi di due cell deve essere al massimo di d_{max}, allora il tasso medio minimo con cui l'attaccante deve inviare cell al relay è di:

$$r_a = \frac{M}{d_{max}} = \frac{M}{Q} \cdot \alpha \frac{B}{n} = \frac{M}{Q} \cdot r. \tag{4.2.3}$$

M/Q è facilmente un numero molto grande vista l'ampia capacità di memoria delle macchine attuali; chi attacca quindi dovrebbe farsi riservare una grande quantità di banda dal router per fare in modo che qualche circuito benigno venga chiuso. Anche se un attacco di questo tipo è possibile in teoria, in pratica risulta molto difficile da condurre, principalmente per due motivi: il primo è che i router servono in modo equo le code dei circuiti, il secondo motivo è che utilizzare quasi tutta la banda del router costituisce da solo un attacco devastante, infatti se l'aggressore disponesse di una larghezza di banda sufficiente a condurre questo tipo di attacco, sceglierebbe un classico attacco DoS che risulterebbe molto più efficace.

4.2.3 Identificazione di servizi nascosti

Per riuscire ad individuare un servizio nascosto, un aggressore deve controllare almeno un relay che può funzionare da guard e almeno un altro router che serve da rendezvous. Un Onion Router, per poter essere etichettato come guard deve soddisfare un uptime minimo e garantire una certo bitrate; mentre per essere utilizzato come punto di rendezvous non ci sono particolari vincoli da rispettare. L'attacco per identificare un servizio nascosto è suddiviso in tre fasi:

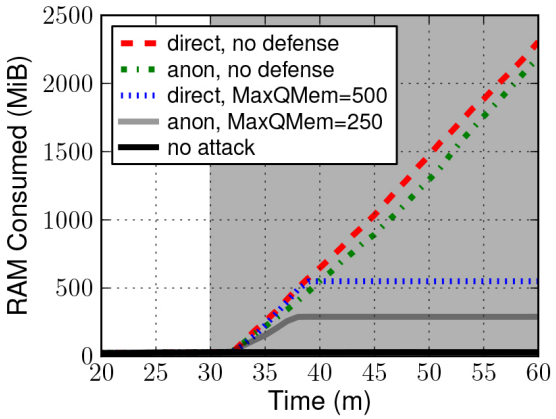

Figura 4.5: Sniper Attack con difesa e senza difesa.

1. identificare i guard del servizio nascosto;
2. disabilitare i guard con lo Sniper Attack;
3. controllare se il server che ospita il servizio nascosto ha selezionato un relay controllato come guard sostitutivo. Se questo non si verifica bisogna ripetere la procedura dal punto 1.

Sia H il servizio nascosto, G_A il relay controllato dall'aggressore che può essere utilizzato come guard, R_A il router controllato da usare come punto di rendezvous e C_A il client/aggressore.

Fase 1

Un client può fare in modo che H stabilisca nuovi circuiti semplicemente inviandogli delle richieste per fissare l'incontro nel punto di rendezvous R_A. Ripetendo molte volte questa operazione il server di H sceglierà molto probabilmente (per qualche circuito) il relay controllato G_A come middle. In questi casi G_A potrebbe potenzialmente sapere quali sono i guard del servizio H, ma non riesce a riconoscerli. Per riuscire ad identificarli il router di rendezvous può inviare un pattern di 50 `PADDING`[5] cell ad H seguiti da una `DESTROY` cell. Se G_A osserva un pattern di 2 cell da H e di 52[6] verso H seguita da una `DESTROY` cell può concludere di essere collegato ad un guard di H. Utilizzando questa tecnica è possibile, in maniera efficiente, identificare tutti i guard di H. Per non creare sospetti è possibile fare le stesse operazioni da C_A (al costo di un rate più basso) con un normale punto di rendezvous, anziché utilizzare sempre lo stesso, includendo delle normali richieste dati per farle sembrare delle richieste ordinarie.

Fase 2

Una volta che tutti i guard di H sono stati identificati, essi vengono disabilitati dall'aggressore attraverso lo Sniper Attack. L'attacco può essere effettuato in parallelo per ridurre il tempo necessario ad escludere dalla rete gli OR; inoltre questo aumenta il tempo in cui tutti i guard sono disabilitati contemporaneamente fornendo un importante vantaggio all'aggressore.

[5] La `PADDING` cell viene utilizzata per indicare che la connessione è ancora attiva, infatti se non vengono scambiati dati, il client e il router inviano l'un l'altro una cell di questo tipo.

[6] Le cell in più sono quelle utilizzate per la costruzione del circuito.

Fase 3

Quando tutti i guard sono stati disabilitati, è sufficiente creare delle normali connessioni con H e determinare con una tecnica simile a quella della fase 1 se il router G_A è stato scelto come relay d'ingresso nella rete. Se questo si verifica è possibile identificare H.

4.2.4 Difesa da attacchi DoS per l'identificazione di servizi nascosti

L'identificazione di servizi nascosti attraverso la tecnica precedentemente descritta, è un operazione che può essere svolta utilizzando qualsiasi attacco DoS da cui la rete Tor è vulnerabile, quindi anche un attacco classico. La vulnerabilità sta nel fatto che un servizio è disposto a scegliere un numero illimitato di entry guard in un certo periodo di tempo; per difendersi quindi è possibile limitare il numero di relay che esso sceglie nei punti cruciali dei propri circuiti, tenendo presente che questo comporta un calo di prestazioni. Inoltre i servizi nascosti rendono troppo semplice l'operazione di identificazione dei guard da parte dell'aggressore, dato che permettono di creare un nuovo circuito ad ogni richiesta di connessione.

La prima cosa da fare è modificare l'algoritmo con cui vengono scelti gli entry guard. Esso cerca di mantenere un certo numero a_g di guard attivi, che corrispondono a quelli che stanno attualmente rispondendo. Inoltre per aumentare la sicurezza fissa un limite r al numero dei guard recenti, cioè che sono stati selezionati meno di t secondi fa. Esso funziona come segue: *se ci sono almeno a_g guard attivi ne viene restituito uno a caso; altrimenti se ci sono meno di r guard recenti, seleziona un nuovo guard fino a quando non tornano disponibili a_g guard attivi o r guard recenti per poi restituirne uno tra questi; altrimenti se ci sono dei guard attivi ne viene restituito uno casualmente; altrimenti ritorna un errore.* L'algoritmo appena descritto permette di limitare il rate con cui vengono aggiunti nuovi relay della lista dei guard.

La seconda modifica consiste nell'utilizzare dei *middle guard*. Un servizio nascosto H deve mantenere un insieme di middle guard $\mathcal{M}_G$ per ognuno degli entry guard. Ogni volta che H sceglie un guard, poi sceglie casualmente un middle guard da $\mathcal{M}_G$ da usare nel circuito. Questo meccanismo di protezione rende più difficile l'individuazione degli entry guard da parte di un potenziale aggressore.

4.3 Packet Spinning

Il *Packet Spinning* [10], come gli altri attacchi DoS, fa in modo che determinati OR non siano più in grado di servire i client che desiderano collegarsi alla rete Tor. Esso utilizza dei loop, dei circuiti artificiali che iniziano e terminano nello stesso punto così da tenere occupati una grande quantità di router alla volta, facendo venire meno la loro capacità di routing. Tipicamente un circuito Tor è costituito da tre router: entry, middle ed exit, ma il protocollo non impone vincoli sulla lunghezza del circuito, permettendo quindi di costruirne di più lunghi. Fintanto che vengono trasmessi dati, il circuito rimane attivo. Inoltre Tor è stato progettato in modo che ogni Onion Router non conosca l'intero percorso del circuito, ma solo, chiaramente, i nodi adiacenti. Il packet spinning si basa su due aspetti fondamentali:

- i circuiti ad anello sono possibili dato che i router non conoscono l'intero path;
- un qualsiasi OR spende una certa quantità di tempo ad effettuare operazioni di tipo crittografico.

Per poter condurre l'attacco è necessario che l'aggressore/client abbia anche sotto il proprio controllo un router R_c.

Per prima cosa l'aggressore deve creare un circuito che inizi e termini in R_c e che passi attraverso tutti gli OR che vuole attaccare. Successivamente il client deve preparare un pacchetto (composto da un certo numero di cell) falso, crittografato livello per livello con le chiavi di sessione negoziate precedentemente e inviarlo al router controllato R_c che la inoltrerà al successivo OR. Ogni router successivo al primo dovrà decifrare il proprio livello e inoltrare il pacchetto al prossimo (come avviene in una normale sessione). Nel momento in cui il pacchetto raggiunge l'ultimo router e viene inviato ad R_c esso lo scarta, reiniettando quello iniziale, crittografato a livelli. Tutta la procedura si può ripetere senza limiti di tempo. Inoltre per rendere più distruttivo l'attacco è possibile costruire più di un loop, combinandoli anche tra di loro, sempre utilizzando R_c come nodo di partenza e di arrivo dato che si occupa del mantenimento dell'anello. Questo attacco risulta particolarmente efficace per tre motivi:

- con un solo router controllato è possibile attaccarne più di uno;
- il router controllato non deve eseguire operazioni crittografiche dato che al termine di ogni giro nell'anello viene reintrodotto il gruppo di cell iniziale;
- la differenza tra il carico di lavoro richiesto al router-aggressore e quella degli altri router aumenta all'aumentare di cell utilizzate.

4.3.1 Rivelazione di comunicazioni anonime

Una volta che l'aggressore ha utilizzato la tecnica precedentemente descritta, può sfruttare questo tipo di attacco per intercettare delle comunicazioni anonime. A questo punto deve utilizzare un gruppo di OR controllati che aveva inserito precedentemente all'interno della rete. Se un client tenta di aggiungere uno dei router sotto attacco per costruire un circuito, molto probabilmente riceverà un timeout dato che sono occupati e non sono in grado di servirlo. Quindi ci sono buone probabilità che i router controllati verranno scelti al posto di quelli benigni. In questo modo possono essere utilizzate delle altre tecniche, ad esempio di timing analysis [11] per individuare i due interlocutori di una conversazione.

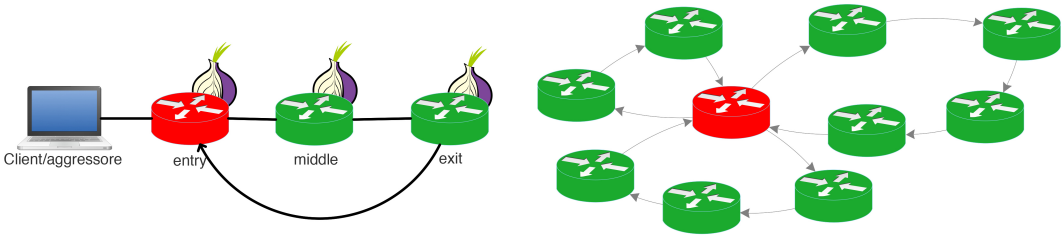

Figura 4.6: Loop singolo a sinistra e loop multipilo a destra.

4.3.2 Difesa attraverso i TBC

Tor è vulnerabile a questo tipo di attacco perché permette di costruire dei loop nella rete. Una metodo molto semplice per risolvere questo problema sarebbe quello di fornire delle informazioni sul circuito a tutti i suoi partecipanti. Così facendo i router si accorgerebbero che il circuito inizia e finisce nello stesso punto, e non permetterebbero il suo completamento. In questo modo, però, il livello di anonimato calerebbe drasticamente visto che i router avrebbero delle informazioni sugli altri partecipanti e tutto questo rischierebbe di rendere la rete troppo insicura e vulnerabile.

Gli autori di [10] propongono di utilizzare i TBC *(Tree Based Circuits)*. Invece di avere un singolo circuito, che si espande telescopicamente dal primo all'ultimo router, è possibile

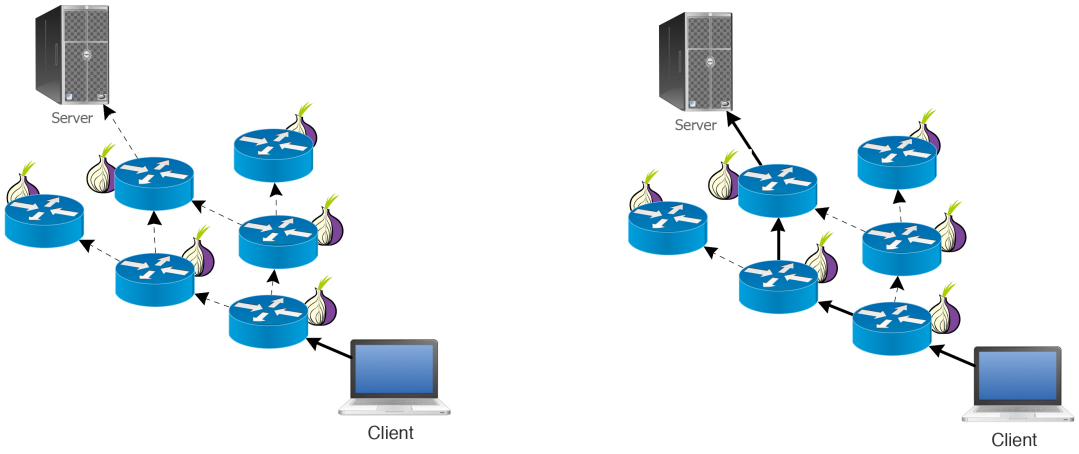

Figura 4.7: TBC, soluzione proposta contro i loop nella rete.

fare in modo di partire dall'entry router, che successivamente stabilirà una connessione con altri due middle router, e ognuno di questi middle router con due exit. Come si può vedere anche dalla Figura 4.7 i circuiti assumeranno quindi una struttura ad albero. Inoltre durante la costruzione dell'albero alcuni OR posso essere scelti in comune. Quando si costruiscono circuiti lunghi, il livello di segretezza aumenta, ma allo stesso tempo anche la latenza. Usando i TBC è possibile aumentare il numero di router che si possono utilizzare, senza aumentare la latenza rispetto ad un circuito normale, visto che la profondità dell'albero può rimanere la stessa. Contrariamente ai normali path, i router devono mantenere delle informazioni aggiuntive sul TBC per poter instradare correttamente le richieste. Inoltre una caratteristica fondamentale dei TBC è l'assenza di loop nella rete: ogni volta che viene scambiata una cell i router possono decidere su quale connessione instradarla e quindi questo evita la presenza di anelli. I TBC offrono più percorsi alternativi verso la destinazione finale e quindi diminuiscono notevolmente le probabilità di attraversare router che sono soggetti ad attacchi di packet spinning. Diminuiscono ma non escludono totalmente dato che comunque un aggressore potrebbe avere sotto il proprio controllo uno o più router e inoltrare le richieste all'entry in modo da creare dei loop artificiali.

5 Conclusioni

In questa tesi sono stati presentati i più recenti ed efficaci attacchi di tipo DoS che possono essere usati nella rete di anonimato Tor. Sono stati inoltre discussi dei possibili metodi di difesa per ogni tipo di attacco. Ogni tecnica di difesa cerca di risolvere il relativo attacco e contemporaneamente di non modificare in maniera radicale il funzionamento del protocollo. Attacchi come il blanket blocking non causano malfunzionamenti della rete, contrariamente al CellFlood Attack o allo Sniper Attack che possono causare un calo di prestazioni della stessa e conseguentemente una diminuzione del livello di sicurezza. Come è stato discusso, il primo risulta aggirabile abbastanza facilmente con l'uso dei bridge o dei tool che permettono di offuscare il protocollo Tor, mentre gli attacchi con un obiettivo specifico richiedono tecniche di difesa più avanzate, come client puzzle e gli altri metodi visti nel capitolo 4. Inoltre il blanket blocking va a colpire una cerchia più o meno ristretta di utenti, mentre gli altri attacchi possono avere conseguenze anche su scala globale. Recentemente non sono state scoperte nuove vulnerabilità nel protocollo che che possono essere sfruttate per condurre attacchi di questo tipo, ma non è da escludere la possibilità che ne vengano individuate di nuove in futuro.

Bibliografia

[1] R. Dingledine, N. Mathewson, and P. Syverson, *"TOR: The second-generation onion router,"* In USENIX Security Symposium (San Diego, CA, 2004), USENIX Association, pp. 303-320.

[2] Tor Metrics https://metrics.torproject.org/, [Accesso: 13 Settembre 2016].

[3] Tor's protocol specifications https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt, [Accesso: 13 Settembre 2016].

[4] Z. Ling, J. Luo, W. Yu, M. Yang, and X. Fu, *"Extensive analysis and large-scale empirical evaluation of Tor bridge discovery,"* in Proc. IEEE INFOCOM, 2012, pp. 2381-2389.

[5] P. Winter, S. Lindskog, *"How the Great Firewall of China is Blocking Tor,"* Free and Open Communications on the Internet, 2012.

[6] The Tor Project. Meek https://trac.torproject.org/projects/tor/wiki/doc/meek, [Accesso: 13 Settembre 2016].

[7] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, *"Blocking-resistant communication through domain fronting,"* Proceedings on Privacy Enhancing Technologies 2015, pp. 1-19.

[8] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. Keromytis, *"CellFlood: Attacking Tor Onion Routers on the Cheap,"* in Proc. ESORICS, Sep. 2013, pp. 664-681.

[9] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, *"The sniper attack: Anonymously deanonymizing and disabling the Tor network,"* in Proc. 21st Annu, Symp. NDSS, Feb. 2014, pp. 1-15.

[10] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos, *"Compromising Anonymity Using Packet Spinning,"* in ISC 08, Sep. 2008.

[11] Lasse Øverlier and Paul Syverson, *"Locating Hidden Servers,"* in Proceedings of the IEEE Security and Privacy Symposium (S&P), May 2006.

[12] E. Erdin, C. Zachor, and M. H. Gunes, *"How to Find Hidden Users: A Survey of Attacks on Anonymity Networks,"*. IEEE Commun. Surv. Tutorials, vol. 17, no. 4, pp. 2296-2316, 2015.

Elenco delle figure

Elenco delle tabelle