POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Infrastruttura per la mitigazione di attacchi DoS basata su un protocollo di puzzle dinamico

Relatori
prof. Luis Mengual Galán
prof. Antonio Lioy

Candidato
Simone Cantarella

Anno accademico 2019-2020

Ai miei genitori

Sommario

La rapida diffusione dell'utilizzo del World Wide Web ha creato cospicui e significativi cambiamenti nella nostra vita quotidiana. L'interruzione della disponibilità di un servizio Internet può comportare gravi danni in diversi ambiti, per questo è necessario essere in grado di prevenire questo tipo di situazione.

Gli attacchi che mirano a bloccare la disponibilità di sistemi o servizi informatici sono generalmente definiti *attacchi denial-of-service* (attacchi DoS). Nell'informatica, un attacco denial of service è un attacco informatico in cui l'autore cerca di rendere un sistema o una risorsa di rete non disponibile ai suoi utenti interrompendo temporaneamente o indefinitamente i servizi offerti da un host connesso a Internet.

L'obiettivo di questa tesi è l'analisi, la progettazione e l'implementazione di un *protocollo di puzzle* in un sistema client-server. In un protocollo di puzzle, un client è tenuto a risolvere un problema computazionale di difficoltà variabile denominato puzzle e deve presentare la soluzione come *proof of work* prima che la sua richiesta venga gestita dal server. Questo tipo di protocollo consente di prevenire congestioni del server e di evitare il consumo di risorse quando il server riceve un gran numero di richieste, imponendo ai client di risolvere un problema basato su funzioni di hash.

Nel sistema sviluppato il server riconoscerà i client dannosi controllando il tasso di arrivo delle richieste e tarderà il loro accesso costringendoli a risolvere un puzzle, mentre i client considerati legittimi saranno in grado di accedere alle risorse ricercate senza ulteriori calcoli e senza subire gli effetti di un attacco in corso.

Per valutare le prestazioni del server, verrà creato un grande numero di client che cercheranno di accedere al server e i risultati della simulazione dimostreranno l'efficienza della soluzione proposta.

Ringraziamenti

Vorrei dedicare questo spazio a chi, con dedizione e pazienza, ha contribuito alla realizzazione di questo elaborato.

Un ringraziamento particolare va al mio relatore esterno Luis Mengual Galán che mi ha seguito, con la sua infinita disponibilità, in ogni step della realizzazione dell'elaborato, fin dalla scelta dell'argomento.

Grazie anche al mio relatore interno Antonio Lioy per i suoi consigli e per avermi suggerito puntualmente le giuste modifiche da apportare alla mia tesi.

Voglio esprimere la mia più profonda gratitudine ai miei genitori ed a tutte le persone che ho incontrato durante i miei studi per avermi fornito un sostegno incessante e un incoraggiamento continuo in questi anni. Questo risultato non sarebbe stato possibile senza di loro. Grazie a tutti.

Indice

Elenco delle figure

Capitolo 1

Introduzione

La rapida diffusione dell'utilizzo del World Wide Web ha creato cospicui e significativi cambiamenti nella nostra vita quotidiana. L'interruzione della disponibilità di un servizio Internet può comportare gravi danni in diversi contesti, come nel caso di un attacco diretto a Dyn, un noto DNS provider, avvenuto nell'ottobre 2016. Questo attacco ha avuto grandi ripercussioni per il mondo digitale e ha creato interruzioni di servizio in molti tra i siti di maggiore spessore mondiale, come AirBnB, Netflix, PayPal, Amazon, Reddit e GitHub.

Gli attacchi che mirano a bloccare la disponibilità di sistemi o servizi informatici sono generalmente definiti *attacchi denial-of-service* (abbreviato **attacchi DoS**). Nell'informatica, il **denial of service** è un attacco informatico in cui l'autore cerca di rendere un sistema o una risorsa di rete non disponibile ai suoi utilizzatori interrompendo temporaneamente o indefinitamente i servizi offerti da un host connesso a Internet. La negazione del servizio viene generalmente ottenuta inondando la macchina o la risorsa obiettivo con un numero elevato di richieste nel tentativo di sovraccaricare il sistema e impedire che alcune o tutte le richieste provenienti da host considerati legittimi vengano soddisfatte.

Poiché sempre più servizi essenziali utilizzano Internet come mezzo della loro infrastruttura di comunicazione, le conseguenze degli attacchi DoS possono essere molto dannose. Quando Internet è stato originariamente sviluppato, il suo obiettivo originale era quello di fornire una rete aperta e scalabile tra le comunità di ricerca ed educazione; per questo motivo, in principio, non sono stati presi in considerazione problemi di sicurezza. Poiché il suo uso è aumentato rapidamente in quasi ogni aspetto della nostra vita, è fondamentale sapersi difendere da queste situazioni potenzialmente dannose.

Per comprendere quanto sia importante evitare questo tipo di attacchi, si possono osservare i dati rilasciati dalla società di sicurezza informatica Imperva [2], che riportano un'analisi statistica di 3.643 attacchi DoS a livello rete nel 2019 e 42.390 attacchi DoS a livello applicazione subiti dai suoi clienti da maggio a dicembre 2019. L'anno 2019 ha visto i più grandi attacchi a livello rete e applicazione mai registrati, con un attacco DoS a livello rete che ha raggiunto la quota di 580 milioni *pacchetti al secondo* [1] (PPS) ad aprile e un attacco a livello applicazione

[1] Gli attacchi DoS sono generalmente misurati dalla quantità di larghezza di banda coinvolta.

durato 13 giorni che ha raggiunto il picco di richieste ricevute con 292.000 richieste al secondo (RPS).

Prevenire gli attacchi DoS può essere molto impegnativo e richiedere molti sforzi, in quanto essi possono verificarsi anche in assenza di vulnerabilità all'interno di un sistema. Nello stesso tempo, è estremamente difficile, se non impossibile, differenziare con precisione tutte le richieste provenienti da un attaccante rispetto a quelle ritenute proveniente da utenti legittimi. Pertanto, le soluzioni che si basano sul rilevamento e sul filtraggio delle richieste in arrivo sui server hanno un'efficacia limitata.

Lo scopo principale di questa tesi è di analizzare il problema del denial of service, concentrandosi sulla mitigazione degli attacchi DoS a livello applicazione e sulla progettazione, implementazione e valutazione di un framework di difesa.

1.1 Disponibilità

L'intento principale di un attacco DoS è quello di insidiare la disponibilità di un servizio. Nella sicurezza informatica, la *disponibilità*, insieme alla confidenzialità e all'integrità, fa parte della triade della CIA (acronimo di Confidentiality, Integrity, Availability), un modello di sicurezza ampiamente utilizzato nelle politiche di difesa dei sistemi informatici per proteggere i dati da accessi non autorizzati e da una possibile loro estrapolazione non concessa. In particolare, la disponibilità si riferisce alla possibilità da parte degli utenti di accedere liberamente a sistemi, reti e dati in qualsiasi momento.

Un fattore molto importante che bisogna tenere in conto quando si cerca di difendere la disponibilità di un servizio è il tempo. Se un servizio non è disponibile per i suoi utenti per un periodo di tempo eccessivo, che supera in modo significativo il tempo di attesa previsto per tale servizio, esso non può soddisfare le richieste dei suoi clienti in modo efficiente, quindi la sua disponibilità viene considerata compromessa. Per questo motivo, quando ci si occupa della disponibilità di un servizio e di una possibile soluzione DoS, bisogna prendere in considerazione anche il tempo richiesto per ottenere l'accesso ad esso. Per questo, una definizione che riassume tale concetto è la seguente:

> *un attacco Denial of Service è l'azione di un utente dannoso perrendere un servizio non disponibile ai suoi utenti per un periodo di tempo eccessivo, in genere, un tempo che supera notevolmente il tempo di attesa previsto.*

1.2 Tipi di attacchi DoS

Esistono tre tipi base di attacchi DoS [4]: 1) *consumo di risorse*, limitate, scarse o non rigenerabili, 2) *distruzione* o alterazione di informazioni di configurazione, 3)

Un'altra misura per descrivere la dimensione di un attacco DoS è il numero assoluto di pacchetti inviati a una rete o a un server web.

distruzione fisica o alterazione dei componenti di rete. In questa tesi, l'attenzione si concentra sull'affrontare il primo tipo di attacchi, vale a dire gli attacchi che consumano le risorse considerate non rinnovabili di un sistema, in particolare quelle che si tenteranno di salvaguardare maggiormente sono la capacità di elaborazione (CPU) e la memoria.

Oltre a questi tre tipi di base, gli attacchi DoS possono essere classificati in varie categorie in base a criteri diversi. La figura 1.1 propone una tassonomia degli attacchi DoS.

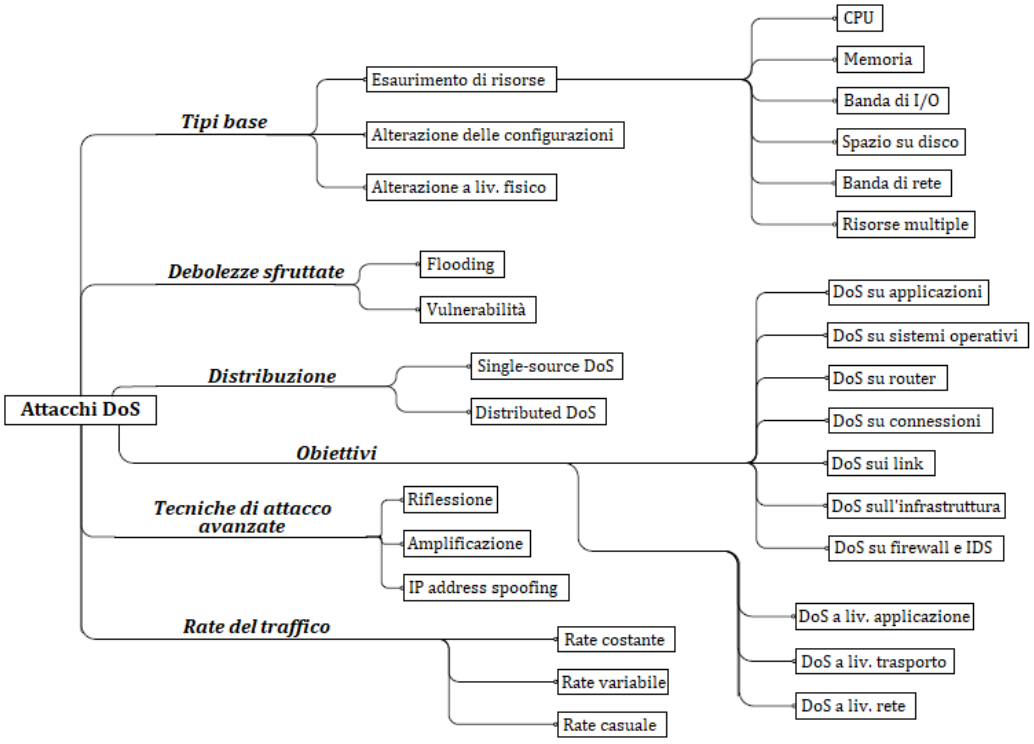

Figura 1.1. Tassonomia degli attacchi DoS

La vittima di un attacco DoS può essere un end-system, un router, una comunicazione in corso, un link o un'intera rete, un'infrastruttura o una qualsiasi combinazionedi questi. Nel caso di un end-system, la vittima può essere un sistema operativo un'applicazione.

A causa del gran numero di categorie, segue una breve discussione sulle categorie più rilevanti per lo sviluppo di questo progetto.

1.2.1 Vulnerabilità e attacchi basati sul flooding

A seconda delle debolezze sfruttate durante un attacco, i diversi tipi di attacchi DoS possono essere classificati in *attacchi basati sulla vulnerabilità* e *attacchi basati sul flooding*.

Un attacco di vulnerabilità DoS sfrutta uno o più difetti nelle politiche di difesa, o bug nel software presente nel sistema obiettivo e mira a consumare una grande

quantità di risorse della vittima. Ad esempio, nell'attacco Ping-of-Death (PoD), un utente malintenzionato può causare, attraverso buffer-overflow, l'arresto anomalo o il riavvio dei sistemi operativi inviando pacchetti ICMP (Internet Control Message Protocol) di dimensioni oltre i 64 KB, ovvero la massima dimensione di un pacchetto IP.

Un attacco flooding, d'altra parte, mira a negare il servizio agli utenti legittimi inviando una grande quantità di richieste di servizio apparentemente affidabili e cercando così di esaurire le risorse della vittima. Ad esempio, in un attacco flooding UDP (User Datagram Protocol), un utente malintenzionato invia un numero eccessivamente elevato di segmenti UDP a porte casuali su un host di destinazione per saturare la sua larghezza di banda, rendendo l'host irraggiungibile da parte degli altri utenti.

Anche in uno scenario in cui tutte le vulnerabilità software e i difetti delle politiche di difesa vengono eliminati, gli attacchi DoS basati sul flooding possono comunque aver luogo. Tuttavia, per essere efficaci, il volume delle richieste di servizio inviate deve essere abbastanza grande da poter saturare le risorse del servizio. Spesso, è difficile per gli aggressori inondare il server inviando un flusso di dati da una sola macchina, poiché i server hanno generalmente maggiori capacità rispetto ai client. Anche se un utente malintenzionato controllasse un client con lo stesso livello di risorse del server di destinazione, l'invio di richieste a una velocità che satura le piene capacità del server potrebbe essere facilmente rilevato a causa del rate d'arrivo molto elevato a cui il server è sottoposto. Pertanto, gli attacchi basati sul flooding vengono in genere eseguiti da un gran numero di macchine distribuite su Internet controllate dagli attacanti. La prossima classificazione degli attacchi DoS riguarda la distribuzione della fonte di attacco.

1.2.2 Attacchi single-source e distribuiti

In un attacco denial of service, gli aggressori possono lanciare i loro attacchi da un singolo host o da più host controllati. Quando i messaggi di attacco sono originati da più host (chiamati *zombies*) distribuiti in rete, l'attacco prende il nome di attacco *Distributed Denial of Service (DDoS)*. Al contrario, quando i messaggi dell'attaccante sono generati da un singolo host, prnede il nome di attacco *Single-Source Denial of Service (SDoS)*.

In generale, gli attacchi DDoS sono più potenti degli attacchi SDoS, poiché la quantità di potenza di elaborazione, memoria e larghezza di banda di più computer attaccanti (che possono essere tra le centinaia o migliaia di macchine) supera di gran lunga le risorse di una singola fonte d'attacco. In pratica, difendersi dagli attacchi DDoS è molto più difficile che difendersi dagli attacchi SDoS.

1.2.3 Attacchi DoS a livello applicazione

Gli attacchi DoS a livello applicazione, detti anche attacchi DoS a livello servizio, mirano a rendere un servizio a livello applicazione non disponibile per i suoi utenti

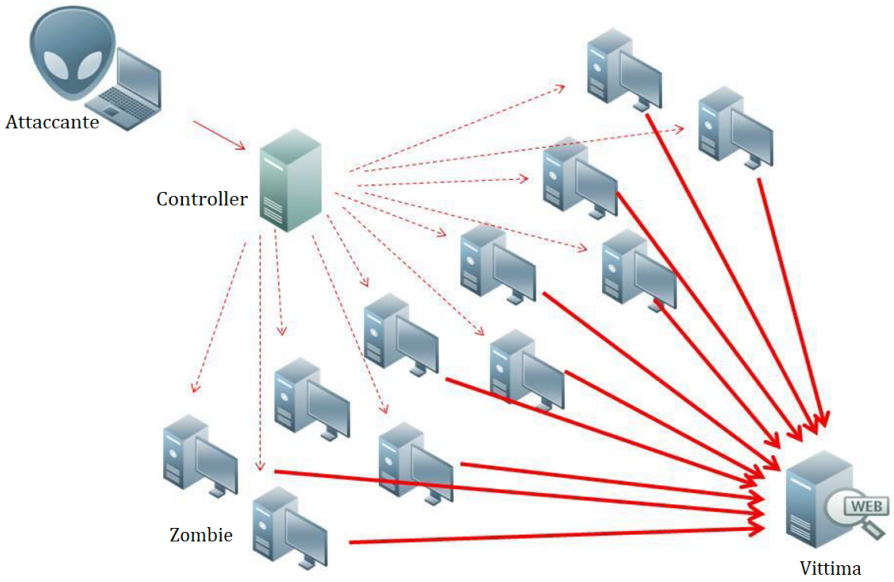

Figura 1.2. Distributed Denial of Service

esaurendo una o più risorse chiave del servizio stesso. Le risorse a cui si fa comunemente riferimento sono potenza di elaborazione (CPU), memoria, larghezza di banda I/O e larghezza di banda della rete.

Attacchi come HTTP flooding, SIP (Session Initiation Protocol) flooding, DNS flooding sono alcuni degli esempi di attacchi DoS a livello applicazione. Un esempio recente di attacco a livello applicazione è stato l'attacco *Mirai* avvenuto nel 2017 ai server DNS di Dyn, che causò gravi interruzioni di servizio nella rete Internet; in quel caso fu creata una botnet che utilizzava dispositivi provenienti da Internet per sfruttare l'attacco.

In passato, molti attacchi DoS hanno preso di mira la larghezza di banda della rete dei sistemi Internet. Tuttavia, con la crescente complessità computazionale delle applicazioni Internet e con una maggiore larghezza di banda di rete disponibile, le risorse dei server come CPU, memoria o larghezza di banda I/O sono diventate il vero collo di bottiglia nella sicurezza informatica. Soprattutto per le applicazioni Web, il collo di bottiglia nell'accesso a un servizio si sta spostando dalla larghezza di banda della rete alle risorse di elaborazione disponibili. Secondo il Global DDoS Threat Landscape Report del primo trimestre 2017 di Imperva [1], gli attacchi DDoS a livello applicazione hanno raggiunto il massimo storico di 1.099 attacchi alla settimana nei mesi tra gennaio e marzo, con un aumento del 23% rispetto agli 892 del trimestre precedente.

La maggior parte dei lavori di ricerca precedenti si sono concentrati principalmente sul combattere gli attacchi DoS a livello rete lasciando in secondo luogo quelli a livello applicazione. Uno dei motivi principali alla base della scarsa preoccupazione a livello applicazione è la rarità del verificarsi di attacchi DoS a questo livello rispetto ad attacchi flooding a livello rete. Tuttavia, questi stanno aumentando in termini di dimensioni, precisione e impatto finale, secondo i recenti rapporti sulle

minacce di Imperva, secondo i quali sono quasi raddoppiati nell'ultimo trimestre del 2017, con il 63,3% degli obiettivi sottoposti a ripetuti attacchi.

Gli attacchi DDoS a livello applicazione richiedono generalmente una larghezza di banda molto inferiore rispetto a quelli di altri livelli per essere efficaci, rendendoli una strategia più facile da mettere in atto e richiedendo meno risorse. Infatti, i vari zombie distribuiti in rete attaccano il server vittima attraverso pacchetti che sembrano avere un formato potenzialmente non dannoso e li inviano sulle normali connessioni TCP, emulando le caratteristiche del traffico a livello rete delle richieste di servizio legittime, rendendo gli attacchi molto più difficile da rilevare.

1.3 Vittime DoS

La vittima designata come bersaglio di un attacco DoS può essere un end-system, un router, una connessione, un collegamento, un'infrastruttura di rete, un'intera rete o una qualsiasi combinazione di questi. Nel caso di un end-system, la vittima designata può essere un sistema operativo o un'applicazione[2]. La presentazione segue illustrando i vari obiettivi (applicazioni, router, ecc) cui un attacco DoS può mirare.

DoS su applicazioni

Negli attacchi DoS alle applicazioni, un utente malintenzionato tenta di impedire all'applicazione di eseguire le sue attività previste facendo sì che l'applicazione esaurisca le capacità di una risorsa specifica. Ad esempio, in ambito eXtensible Markup Language (XML), nella tipologia d'attacco chiamato Exponential Entity Expansion (noto anche come attacco Billion Laughs), un utente malintenzionato passa a un parser XML un piccolo documento XML che sembra essere sia ben formato che valido (queste sono caratteristiche per descriver un documento XML), ma successivamente questo si espande in un file molto grande. Quando il parser tenta di analizzare l'XML, questo finisce per consumare tutta la memoria disponibile all'applicazione del parser dovuto alle grandi dimensioni del file.

Di solito, le risorse riservate alle applicazioni sono vincolate dalla configurazione del sistema su cui vengono eseguite, come ad esempio il numero massimo di processi o thread dedicati, o il numero massimo di connessioni simultanee che esse possono creare, allo scopo di limitare l'impatto di un possibile overhead sull'intero sistema operativo. Tuttavia, se tali limiti non vengono scelti con cura in base al ruolo destinato alla macchina su cui si trovano (ad esempio, un server Web ha bisogno di più risorse di un personal computer), importanti applicazioni possono diventare facili obiettivi DoS.

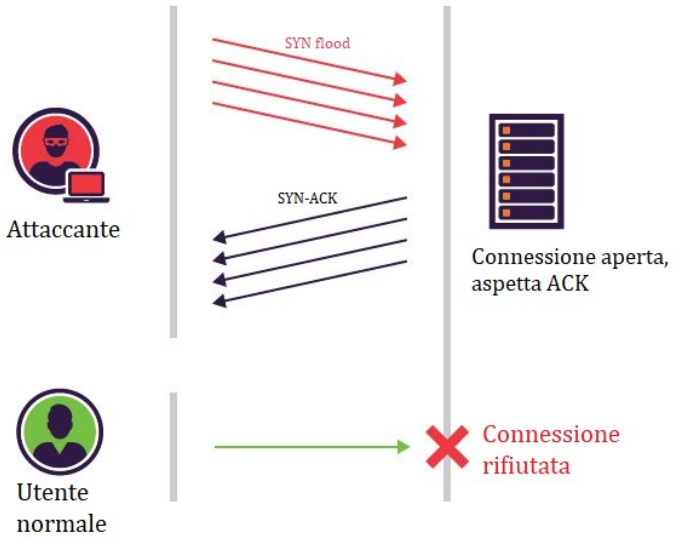

Figura 1.3. Attacco SYN flood

DoS su sistemi operativi

Gli attacchi DoS ai sistemi operativi sono molto simili agli attacchi DoS alle applicazioni. Tuttavia, negli attacchi DoS alle applicazioni, il sistema operativo potrebbe essere in grado di proteggere le altre applicazioni dagli effetti dell'attacco; è quindi ovvio quanto un attacco DoS a un sistema operativo possa essere molto più pericoloso. Un attacco molto noto a un sistema operativo è quello del SYN flooding (rappresentato in figura 1.3) che avviene utilizzando il Transmission Control Protocol (TCP), in cui un utente malintenzionato invia un flusso di pacchetti TCP SYN alla vittima senza completare l'handshake tipico del TCP e quindi riesce ad esaurire la memoria della vittima lasciando aperte connessioni non utilizzate. Un simile attacco ha effetto su tutte le applicazioni che utilizzano TCP come protocollo di comunicazione, rendendo quindi il sistema operativo compromesso.

DoS su router

Molti degli attacchi DoS contro un end-system possono anche essere lanciati contro un router IP. Infatti, i protocolli di routing possono essere utilizzati, oltre al loro scopo principale, anche per creare un attacco DoS su un router o una rete di router. Ciò richiede la capacità di poter inviare traffico da indirizzi da cui ci si aspetta di ricevere messaggi di routing. L'attacco più semplice a un router è quello di sovraccaricare la sua tabella di routing con un numero sufficientemente grande di rotte, facendo in modo che il router esaurisca la propria memoria, oppure facendo in modo che esso esaurisca la CPU disponibile per elaborare le numerose rotte ricevute. Attacchi DoS più gravi su router utilizzano aggiornamenti di percorsi falsi che possono causare il black-holing di un intero blocco di indirizzi di rete.

[2]Il termine end-system corrisponde qui al termine "host Internet", dove un host è un computer che implementa tutti i livelli dello stack TCP/IP.

DoS sulle connessioni

Invece di attaccare l'end-system, un attaccante può tentare di interrompere una connessione in corso. Se un utente malintenzionato è in grado di intromettersi in una connessione TCP, è relativamente semplice falsificare i pacchetti per reimpostare tale connessione o per sincronizzarla in modo tale da non poter compiere ulteriori operazioni. Anche se un utente malintenzionato non è in grado di intervenire in una connessione TCP, esso può scoprire l'esistenza di tale connessione, e quindi può ripristinare o sincronizzare tale connessione inviando un numero elevato di pacchetti di ripristino TCP falsi che cercano di scoprire i numeri di porta TCP e i sequence number della connessione.

DoS sui link

La forma più semplice di attacco DoS sui link è l'invio di traffico per cui non venga controllata la presenza di congestione (ad es. traffico UDP), in modo tale che il collegamento si congestioni debito all'elevato numero di dati inviati e il traffico normale subisca una perdita elevata di pacchetti. La congestione di un collegamento potrebbe anche causare problemi sui protocolli di routing, nel caso in cui si verificassero perdite di pacchetti appartenenti a protocolli di routing. Inoltre, può essere possibile negare l'accesso a un collegamento facendo in modo che un router invii una gran quantità di traffico di configurazione cosicché possa sovraccaricare i link su cui si trovano le sue porte. I messaggi del protocollo SNMP (Simple Network Management Protocol) sono un possibile mezzo per questo attacco, in quanto su di essi non vengono normalmente effettuati controlli sulla congestione.

DoS sull'infrastruttura

Molti sistemi di comunicazione dipendono da alcune infrastrutture sottostanti per le loro normali operazioni. Una detta infrastruttura può avere la grandezza di un domain name system globale o può essere piccola come una rete Ethernet locale o un punto di accesso wireless. Gli attacchi alle infrastrutture possono avere enormi effetti sui loro utenti. Ad esempio, il protocollo Domain Name System (DNS) funge come una rubrica per l'intero Internet traducendo nomi di host in indirizzi IP. Negare l'accesso a un server DNS nega quindi l'accesso a tutti i servizi come il Web, e-mail, chiavi pubbliche, certificati, ecc., che vengono serviti da quel server DNS. Maggiore è la zona di cui è responsabile un server DNS, maggiore è l'impatto di un attacco DoS. Per esempio, nel 2002, tutti i tredici server DNS root sono stati sottoposti a un attacco DoS su larga scala. Per questo, alcuni root nameserver erano irraggiungibili da molte parti della rete globale.

Alcuni attacchi DoS prendono di mira un'infrastruttura utilizzata congiuntamente da tutti gli host in una rete locale. Ad esempio, un attacco tenta di accedere a una sotto-rete potrebbe essere in grado di impedire ad altri host di accedere alla rete esaurendo il blocco di indirizzi allocato da un server DHCP (Dynamic Host Configuration Protocol). Sebbene tali attacchi richiedano la possibilità di falsificare l'indirizzo MAC di una scheda Ethernet o wireless, sono comunque realizzabili con determinati hardware e sistemi operativi.

DoS su firewall e IDS

I firewall hanno lo scopo di difendere i sistemi alle loro spalle contro minacce esterne limitando il traffico dati da e verso i sistemi protetti. I firewall possono anche essere utilizzati per difendersi dagli attacchi DoS. Tuttavia, i firewall stessi possono diventare l'obiettivo di un attacco. I firewall possono essere classificati come stateful e stateless, in base al fatto che il firewall mantenga lo stato per i flussi dati che lo attraversano, dove un *flusso* è uno stream di pacchetti che condividono indirizzi IP sorgente e destinazione, numeri di porta sorgente e destinazione e protocollo utilizzato.

I firewall stateless, in genere, possono essere attaccati tentando di esaurire le risorse di elaborazione del firewall. Oltre all'esaurimento della potenza di elaborazione, i firewall staeful possono anche essere attaccati inviando traffico fittizio facendo in modo che il firewall esaurisca la sua memoria e non possa così più creare istanze per gli stati dei flussi legittimi. Nella maggior parte dei casi ciò causerà negazione del servizio ai sistemi protetti, poiché la maggior parte dei firewall causa la disconnessione dei sistemi alle sue spalle nel caso esso cessi di funzionare correttamente.

I *sistemi di rilevamento delle intrusioni* (IDS, acronimo di *Intrusion Detection System*) presentano problemi simili a quelli dei firewall. A differenza di un firewall, un IDS è normalmente fail-open, il che non negherà il servizio ai sistemi protetti da un IDS. Tuttavia, ciò significa che gli attacchi successivi non verranno più riconosciuti poiché l'IDS ha smesso di funzionare.

1.4 Ambito del problema e obiettivi di design

1.4.1 Ambito del problema

In questa tesi, l'attenzione si concentra sulla mitigazione degli attacchi flooding distribuiti a livello applicazione nei servizi che utilizzano la rete Internet. Per *livello applicazione*, si intente il livello 7 del modello di riferimento OSI, il quale fornisce servizi direttamente agli utenti finali (ad esempio e-mail, servizi di trasferimento file, ecc). L'obiettivo non è quello di affrontare vulnerabilità specifiche in determinate applicazioni o protocolli, bensì quello di fornire un framework più generale che possa essere modificato per una specifica applicazione o servizio. Il principale caso di test in questa tesi sarà un'applicazione che tenta di accedere a delle informazioni conservate in un database.

Con *distribuito*, ci si riferisce agli attacchi DoS distribuiti che sono stati definiti nella sezione precedente.

Con *service flooding*, ci si riferisce agli attacchi DoS basati sul flooding come definiti nella sezione precedente. Gli attacchi basati sulla vulnerabilità non verranno presi in considerazione, poiché in questo caso le richieste di attacco spesso cercano di sfruttare difetti specifici nel sistema della vittima (ad es. buffer overflow), difetti il cui rilevamento è obiettivo di molti sistemi di rilevamento e firewall. Sebbene

possa essere difficile rilevare l'esistenza di determinati pattern nelle richieste dell'attaccante, saperli trovare è comunque un modo per differenziarli dalle richieste legittime. Ad esempio, negli attacchi DDoS basati sul flooding, le richieste di servizio degli attaccanti differiscono dalle richieste legittime solo nell'intento ma non nelle caratteristiche dei pacchetti inviati, rendendo così tali attacchi più difficili da essere rilevati.

In termini di risorse prese di mira dall'attacco, l'obiettivo è quello di proteggere le risorse ritenute essenziali per l'elaborazione di una richiesta di servizio da parte del server. L'elaborazione di una richiesta comporta spesso una combinazione di CPU, memoria, larghezza di banda I/O, ecc; proteggere queste risorse sarà sufficiente per il tipo di attacchi DoS considerati.

1.4.2 Obiettivi di design

Questo paragrafo mostra un comune tipo di difesa DoS basato su un protocollo di puzzle al fine di illustrare gli obiettivi di design di questa tesi.

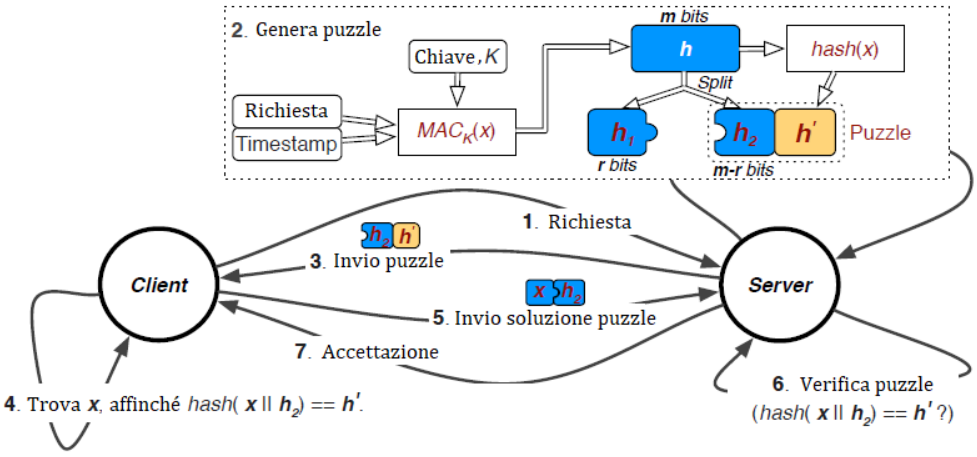

Figura 1.4. Schema di un protocollo di puzzle basato su hash

In un protocollo di puzzle, un client è tenuto a risolvere un problema computazionale di difficoltà variabile denominato puzzle e deve presentare la soluzione come *proof of work* prima che la sua richiesta venga gestita dal server. Ad esempio, in un puzzle basato su hash, il server calcola un Message Authentication Code (MAC) h di m-bit utilizzando il messaggio di richiesta ricevuto dal client e un timestamp come input, ovvero $h = MAC(request||timestamp)$ e calcola l'hash digest h' di h come $h' = hash(h)$, dove $hash(h)$ è una funzione hash crittografica, come ad esempio SHA-1. Il server quindi suddivide h in $h1$ e $h2$, le cui lunghezze sono rispettivamente di r e $m - r$ bit, e invia $h2$ con r insieme all'hash digest h' al client. Il client, quindi, cerca attraverso un metodo brute force un valore x di r bit per cui $hash(x||h2)$ sia uguale a h', e invia x insieme a $h2$ al server come soluzione

del puzzle. Dopo aver ricevuto la soluzione del puzzle presentata dal client, il server può ricalcolare h' senza memorizzarlo e verifica se $hash(x||h2)$ è uguale a h' e concede il servizio al client se la soluzione è corretta.

Nel suddetto protocollo detto *hash reversal* (poichè l'obiettivo è trovare l'input di una funzione hash), il server può aumentare/diminuire la quantità di calcoli che il client deve eseguire aumentando/diminuendo il numero r di bit della parte mancante $h1$ del puzzle. Questa lunghezza viene generalmente definita *difficoltà del puzzle*.

I protocolli di puzzle possono mitigare l'effetto degli attacchi DoS, poiché se un utente malintenzionato invia sempre maggiori richieste ad un server, più puzzle dovrà risolvere e quindi necessiterà maggiori risorse computazionali e ciò reallenterà il suo tasso di invio di richieste. Tuttavia, a causa della variazione della potenza computazionale dei vari client, i client con maggiori risorse computazionali possono risolvere puzzle a un tasso molto più elevato rispetto a client meno potenti, facendo in modo che il server elabori le loro richieste molto più spesso rispetto alle richieste dei client più deboli.

Un altro difetto rilevante dei protocolli di puzzle è che tutti i client, compresi anche tutti quelli legittimi, sono tenuti a eseguire calcoli che richiedono un grande uso della CPU, i quali però non forniscono alcuna utilità senonché quella di ottenere l'accesso al server. Nell'hash reversal puzzle di sopra, ad esempio, la soluzione del puzzle, ottenuta calcolando un gran numero di hash, viene semplicemente eliminata dopo che la sua validità è stata confermata dal server.

Lo studio delle soluzioni di difesa già esistenti basate su puzzle, come quello illustrato in questo paragrafo, ha fornito importanti suggerimenti su quali requisiti debba soddisfare un framework di difesa basato su puzzle per essere efficace contro gli attacchi DoS a livello di applicazione. I principali sono i seguenti:

Nessun affidamento sul rilevamento Il fondamento su cui si basa la maggior parte degli attuali modelli di rilevamento di attacchi DoS è che le caratteristiche del traffico attaccante differiscono da quelle del traffico normale, ma ciò non vale sempre poiché sofisticati attacchi DoS a livello applicazione possono emulare il comportamento delle richieste dei clienti legittimi. Pertanto, il sistema di difesa che si svilupperà dovrà essere in grado di mitigare gli effetti degli attacchi DoS senza fare affidamento su meccanismi di rilevamento.

Uguaglianza per tutti i client La difesa basata su puzzle deve essere indipendente dalla variabile capacità di calcolo dei client. In altre parole, il framework non deve gravare sui client con una potenza di calcolo limitata e deve garantire ad ogni client una giusta parte della capacità del server.

Ridurre il lavoro superfluo Come detto prima, le soluzioni basate su puzzle impongono ai client calcoli che utilizzano molta CPU e momoria, i quali non hanno nessun'altra utilità tranne che quella di garantire l'accesso ai client. Un protocollo di puzzle deve ridurre al minimo questi calcoli dispendiosi.

Modifiche minime al client Affinché una soluzione DoS abbia successo, essa deve essere facilmente adattabile e distribuibile nella pratica. La facile adattabilità richiede una modifica minima ai programmi client e server. In particolare, un client deve essere in grado di interagire con il server senza richiedere modifiche funzionali.

1.5 Organizzazione della tesi

Il resto dei capitoli di questa tesi, con contenuti e scopi, sono organizzati come segue. Il capitolo 2 esamina varie soluzioni di difesa DoS esistenti, con particolare attenzione alla difesa DoS basata su puzzle. Il capitolo 3 introduce l'analisi di un sistema di difesa DoS basato su puzzle in un'architettura client-server. Inoltre, descrive il threat model considerato in fase di sperimentazione. Il capitolo 4 illustra l'implementazione della soluzione proposta. Nel Capitolo 5, viene illustrato il modello sperimentale utilizzato per valutare la soluzione proposta e i risultati ottenuti. Infine, le considerazioni finali e i possibili lavori futuri sono mostrati nel Capitolo 6.

Capitolo 2

Stato dell'arte

Un cospicuo numero di soluzioni è stato proposto in letteratura per gestire il problema del Denial of Service. Lo scopo di questo capitolo è di introdurre le varie soluzioni già esistenti, riassumere le tecniche utilizzate in queste soluzioni e valutarne i punti di forza e di debolezza. L'attenzione è focalizzata sugli esempi più rappresentativi per ciascuna categoria di strategia di difesa.

2.1 Problemi di difesa DoS

Quando Internet è stato sviluppato, non si è tenuto conto della possibilità di attacchi da parte degli utenti di Internet. In quest'ottica, gli strumenti utili per monitorare e prevenire comportamenti dannosi da parte degli utenti non sono mai stati progettati o implementati. La maggior parte dei problemi relativi agli attacchi DoS sono dovuti alla mancanza di sicurezza nell'architettura Internet ideata originalmente, ma altri sono inerenti al problema generale del DoS.

Questi sono i problemi più rilevanti relativi agli attacchi DoS:

Difficoltà nel distinguere le richieste dannose: è difficile distinguere tra richieste dannose e legittime. Questo è valido per pacchetti, flussi di rete, segmenti di livello trasporto o messaggi di richiesta di servizio delle applicazioni. Anche se determinati comportamenti dannosi possono essere rilevati in modo affidabile da meccanismi di rilevamento degli attacchi basati su firma, gli attaccanti possono modificare le caratteristiche dei loro messaggi per eluderne il rilevamento. I rilevamenti diventano più difficili quando il sistema è sottoposto ad attacchi flooding altamente distribuiti, poiché tali attacchi inviano un innumerevole numero di messaggi per sfruttare determinate vulnerabilità. Non è necessario che tali pacchetti negli attacchi flooding abbiano un formato errato (ad esempio, abbino il campo di frammentazione non valido o un payload di pacchetti dannoso) per essere efficaci. Grazie a ciò, gli attaccanti sono in grado di creare messaggi d'attacco che non sono distinguibili dai messaggi di richiesta legittimi. Infine, in linea di principio, non è possibile distinguere tra un attacco DoS e una raffica di richieste legittime in arrivo simultaneamente.

Asimmetria nell'overhead di richieste e risposte: si riferisce all'asimmetria, misurata come quantità di risorse consumate, nel generare una richiesta lato client e creare la sua corrispondente risposta lato server. Nella maggior parte dei casi, un client richiede un consumo minimo di CPU e risorse di memoria per generare una richiesta, mentre le operazioni eseguite dal server per produrre la risposta richiedono un overhead di risorse significativamente maggiore. A peggiorare le cose, gli attaccanti possono creare "off-line" le loro richieste prima dell'attacco, riducendo ulteriormente il sovraccarico dovuto alla generazione di una richiesta di servizio.

Gestione decentralizzata: un importante requisito di progettazione dell'architettura Internet è che deve consentire una gestione distribuita delle sue risorse. L'attuale Internet è un'interconnessione di molti sistemi autonomi (AS), in cui ogni sistema autonomo è un insieme di router e collegamenti stanti sotto un'unica amministrazione tecnica. Ogni sistema autonomo definisce il proprio insieme di criteri operativi e di sicurezza. L'imposizione di una politica di sicurezza globale è enormemente difficile poiché richiede una cooperazione tra i vari AS. D'altra parte, molti attacchi DDoS potrebbero non essere evitati ponendo difese in un unico punto e richiederebbero quindi che i meccanismi di difesa vengano distribuiti in più posizioni di Internet. Progettare soluzioni in grado di soddisfare questi requisiti contrastanti è un compito molto complicato.

Spoofing indirizzo IP[1]: i pacchetti con indirizzi IP contraffatti sono più difficili da filtrare poiché ogni pacchetto falsificato sembra provenire da un indirizzo diverso, nascondendo quindi la vera origine dell'attacco. La proliferazione di botnet di grandi dimensioni rende attualmente lo spoofing meno utilizzato negli attacchi denial of service, ma gli aggressori hanno comunque a disposizione lo spoofing come metodo d'attacco, quindi le difese contro gli attacchi denial of service che si basano sulla validità dell'indirizzo IP sorgente dei pacchetti potrebbero avere problemi con pacchetti IP contraffatti.

Difficoltà nella ricerca sulla difesa DoS: l'avanzamento della ricerca nel campo della mitigazione degli attacchi DoS è stata storicamente ostacolata dalla mancanza di informazioni sugli attacchi, di parametri di valutazione di difesa standard e dalla difficoltà di test su larga scala. Poche informazioni sugli incidenti DoS sono disponibili pubblicamente a causa della riluttanza delle organizzazioni a rivelare il verificarsi di un attacco, per paura di danneggiare la propria reputazione aziendale. Senza un'analisi dettagliata degli attacchi DoS avvenuti realmente, è difficile progettare soluzioni adattabili al mondo reale.

In termini di parametri di valutazione standard, non esiste uno standard per valutare l'efficacia di un sistema di difesa DoS. La mancanza di approcci di valutazione standard spesso porta a una situazione in cui i ricercatori e i progettisti tendono a presentare i risultati dei propri test di valutazione che sono

[1]Nelle reti di computer, lo spoofing dell'indirizzo IP o IP spoofing è la creazione di pacchetti IP con un indirizzo IP sorgente falso, allo scopo di impersonare un altro sistema informatico.

più vantaggiosi per il loro sistema. Inoltre, ciò rende molto difficile confrontare le prestazioni di varie soluzioni. In aggiunta, il testing delle soluzioni DoS in un ambiente realistico è immensamente impegnativo, a causa della mancanza di simulazioni su larga scala o di strumenti di simulazione dettagliati e realistici in grado di simulare reti della dimensione simile a quella di Internet.

2.2 Strategie di difesa

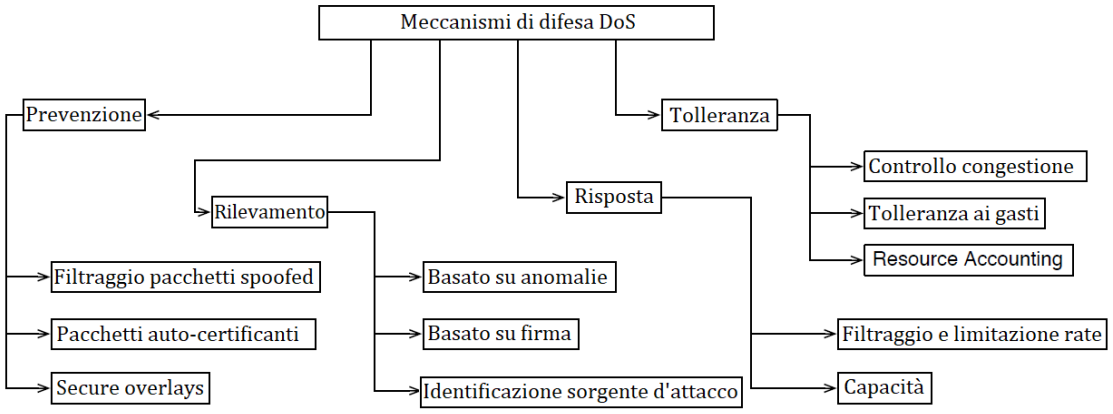

Figura 2.1. Tassonomia dei meccanismi di difesa

Le strategie dei meccanismi di difesa degli attacchi DoS possono essere suddivise in quattro categorie: *prevenzione*, *rilevamento*, *risposta* e *tolleranza*. La *prevenzione* tenta di eliminare il verificarsi di attacchi DoS o di impedire che l'attacco provochi danni significativi. Il *rilevamento* può essere ulteriormente classificato come *rilevamento di attacco* e *rilevamento della sorgente d'attacco*. Il rilevamento di attacco monitora e analizza gli eventi in un sistema per scoprire tentativi malevoli di accesso. È un passo importante prima di eseguire ulteriori azioni per contrastare un attacco. *L'identificazione della sorgente d'attacco*, d'altra parte, mira a trovare la posizione della sorgente di un attacco anche quando l'indirizzo sorgente delle richieste malevole è un indirizzo falso o errato. I meccanismi di *risposta* vengono solitamente avviati dopo il rilevamento di un attacco per eliminare o minimizzare l'impatto dell'attacco sulla vittima. La *tolleranza* mira a minimizzare i danni causati da un attacco DoS senza essere in grado di differenziare gli accessi dannosi da quelli legittimi. In questo caso, può essere sufficiente solamente sapere se il carico computazionale nel sistema da difendere sia al di sopra di una certa soglia al fine di avviare i meccanismi di tolleranza.

Ciascuna delle quattro categorie di difesa precedenti può essere ulteriormente suddivisa in diversi tipi di meccanismi di difesa, in base alle caratteristiche delle diverse soluzioni. La figura 2.1 illustra la tassonomia dei meccanismi di difesa introdotta precedentemente per classificare le soluzioni DoS già esistenti.

Poiché gli attacchi DoS possono essere contrastati in diverse posizioni della rete, i meccanismi di difesa possono anche essere classificati in base alla posizione

di dove vengano applicati i meccanismi di difesa. In Internet sono possibili quattro posizioni per implementare le difese DoS: *vicino alla vittima*, *nella rete intermedia*, *vicino alla fonte d'attacco* e *posizioni multiple*. Nella strategia *vicino alla vittima* i meccanismi di difesa agiscono nella rete a cui appartiene la vittima, contrariamente alla strategia *vicino alla fonte d'attacco* dove agiscono nella rete in cui si trova la sorgente dell'attacco. Nella soluzione *rete intermedia* la protezione viene effettuata sulle reti che si trovano tra quella di origine e quella di destinazione; infine, *posizioni multiple* è un congiunto delle tre soluzioni precedenti. La figura 2.2 illustra un esempio di rete semplificato di Internet, in cui si mostrano le varie posizioni in cui si possono applicare le difese DoS.

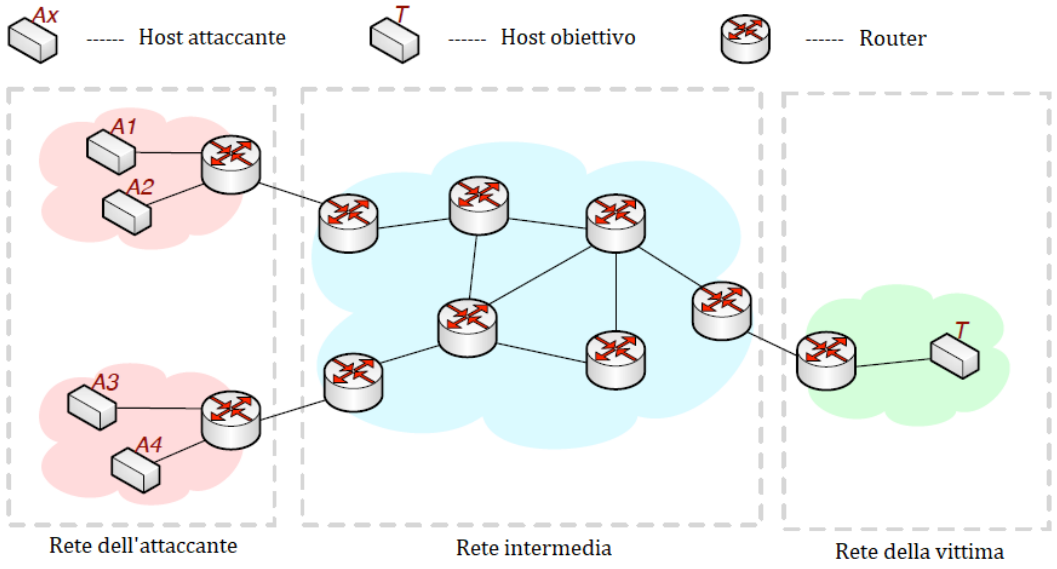

Figura 2.2. Posizione delle difese DoS in una rete semplificata

Lo sviluppo di questa tesi segue i principi del meccanismo di tolleranza, pertanto a seguire saranno introdotte le più importanti soluzioni già esistenti legate a questo tipo di meccanismo di difesa.

2.3 Tolleranza

L'ovvio vantaggio dei meccanismi di tolleranza è che non si basano su meccanismi di rilevamento per identificare la presenza di attacchi e in alcuni casi non hanno nemmeno bisogno di sapere se sia in corso un attacco. Ciò risulta molto utile quando è particolarmente difficile distinguere il traffico malevolo da quello legittimo oppure quando la precisione di rilevamento è bassa. I meccanismi di tolleranza riconoscono l'impraticabilità della prevenzione o del rilevamento preciso di attacchi DoS e si occupano di ridurre al minimo l'impatto degli attacchi e di massimizzare la qualità del servizio offerto durante un attacco.

I meccanismi di difesa che si basano sulla tolleranza possono essere raggruppati approssimativamente in tre categorie: *controllo della congestione*, *tolleranza ai guasti* e *resource accounting*.

2.3.1 Controllo della congestione

Il controllo della congestione nelle reti IP viene in genere eseguito su ciascun router tramite la gestione delle code dei pacchetti e la rete affida agli host finali il compito di reagire alla congestione. Tuttavia, quando raffiche di pacchetti provenienti da sorgenti malevoli continuano a essere inviate a velocità molto elevate, i sistemi di gestione delle code implementati negli attuali router rivelano una carenza significativa nella protezione dei flussi dati provenienti da sorgenti legittime. Due esempi di come i meccanismi di controllo della congestione possono essere in grado di fornire sicurezza sono *Re-feedback* e *NetFence*.

Re-feedback

Il motivo che ha portato alla soluzione Re-feedback [7] è che i meccanismi di controllo della congestione esistenti sono difficili da mettere in pratica, come in Explicit Congestion Notification [2] (abbreviato ECN), in cui si è in grado di rilevare la congestione solamente nell'ultimo nodo di uscita di una rete intermedia, non veramente efficace in quanto le difese nei nodi in ingresso sono più efficienti. I suoi autori hanno proposto di raccogliere informazioni sulla congestione e sul ritardo della rete dai campi dell'intestazione dei pacchetti IP in una maniera receiver-aligned mentre i pacchetti effettuano un determinato percorso. In questo caso, receiver-aligned significa che il valore di un campo di intestazione IP inizia con vari valori diversi nei mittenti, ma si ferma a un valore fisso concordato in precedenza quando il pacchetto raggiunge il destinatario. Ad esempio, il Time to Live (TTL) è sender-aligned in quanto inizia sempre con un valore fisso al mittente (255). Affinché sia receiver-aligned, il TTL dovrebbe arrivare al ricevitore impostato su un valore fisso, ad esempio 0. Per ottenerlo allineato al ricevitore, ogni ricevitore deve comunicare il valore del TTL che vede agli altri utenti, in modo che i mittenti possano capire quale numero inviare per allinearsi al ricevitore.

Gli autori del Re-feedback hanno proposto di utilizzare il TTL e un nonce per implementare la loro soluzione. Quando Re-feedback è in esecuzione, ogni pacchetto arriva a ciascun nodo di rete portando una visione dello stato della congestione a valle del nodo. Di conseguenza, la visione della congestione completa della rete diventa visibile solo nel più vicino alla rete della vittima, dove un rate policer potrebbe essere più utile.

Tuttavia, non vi è alcuna garanzia che un mittente comunichi sempre le informazioni sulla congestione in modo veritiero. Per risolvere questo problema, il Re-feedback mira a creare un ambiente in cui anche comportamenti non affidabili aiutano a una non congestione della rete.

Per garantire che i mittenti impostino le informazioni sulla congestione in modo veritiero, viene utilizzato un dropper nell'ultimo nodo di uscita della rete. Ogni

[2]Explicit Congestion Notification (ECN) è un'estensione dei protocolli IP e TCP, definito nella RFC 3168 (2001). Questa estensione consente la notificazione end-to-end della presenza di congestione nella rete evitando l'eliminazione dei pacchetti.

utente assegna un valore sul valore della congestione percepito. Se un mittente sottostima tale valore, il valore della congestione assume un valore quando il pacchetto raggiunge il dropper in uscita e di conseguenza il dropper lo scarta. Lo schema prevede anche che sia nell'interesse dell'operatore di rete sorvegliare e rispondere alla congestione degli utenti. Quindi, se un mittente sovrastima la congestione, il policer di ingresso della propria rete limiterà il traffico proveniente da quell'utente. Successivamente, vengono utilizzati meccanismi di "addebito" tra sotto-reti per garantire che qualsiasi rete che ospita host "zombie" debba pagare per la congestione causata dagli utenti presenti nella propria rete. Con questo metodo di addebito, il valore attribuito una rete dipende dalla differenza tra la congestione al suo ingresso (le sue entrate) e alla sua uscita (il suo costo). Quindi sovrastimare la congestione interna aumenta il valore della rete. Considerando che il routing organizza i suoi percorsi verso le rotte meno costose, questo metodo fa sì che una rete che falsifica le statistiche sulla congestione rischi di perdere le rotte verso reti a costi migliori.

Se il meccanismo di re-feedback è in esecuzione, esso può aiutare ad alleviare la congestione della rete e l'efficacia degli attacchi DoS. Tuttavia, non è chiaro se Re-feedback possa aiutare ad alleviare attacchi DoS su larga scala. Altri svantaggi del Re-feedback sono i seguenti: in primo luogo, il policer all'inizio della rete deve mantenere lo stato per ogni flusso dati al fine di controllare la risposta alla congestione dei mittenti, il che espone il policer stesso ad attacchi DoS per via dell'esaurimento delle sue risorse. In secondo luogo, è difficile per il mittente impostare correttamente il vero valore di congestione, poiché potrebbe richiedere più verifiche per accertarne la validità. A peggiorare le cose, l'invio di un valore di congestione sottovalutato aumenta il rischio di scarto al dropper di uscita, mentre l'invio di un valore di congestione sovrastimato aumenta il rischio di sanzioni al policer di ingresso. Terzo, se viene eseguito un attacco DDoS che sottostima la congestione, il traffico attaccante può essere eliminato solo nell'ultimo punto di uscita vicino al ricevitore. Il traffico di attacco consuma quindi ancora una grande quantità di larghezza di banda sul percorso verso la destinazione e può influire in modo significativo su altri mittenti che condividono lo stesso percorso. Ultimo, ma non meno importante, Re-feedback funziona bene per i protocolli simili a TCP con un meccanismo di riconoscimento (ACK), ma non funziona per i protocolli, come UDP, che non dispongono di tali meccanismi.

NetFence

NetFence [16] è un'architettura di rete resistente agli attacchi DoS che utilizza un meccanismo di feedback sicuro per il controllo della congestione all'interno di una rete. I router in congestione inseriscono nei pacchetti un feedback di controllo della congestione non falsificabile e i router di accesso della rete esaminano il valore dei feedback, sorvegliando allo stesso tempo il traffico degli utenti. Il feedback di congestione può essere utilizzato, dai ricevitori, per controllare il traffico indesiderato. NetFence considera ogni Autonomous System come unità di rete e propone di utilizzare l'accodamento a livello AS e la bassa velocità di scambio ai confini tra gli AS per limitare i danni DoS nei sistemi autonomi che ospitano router in congestione.

NetFence distingue i pacchetti in: pacchetti di richieste, pacchetti regolari e pacchetti legacy; inoltre richiede che ogni router mantenga tre code diverse per

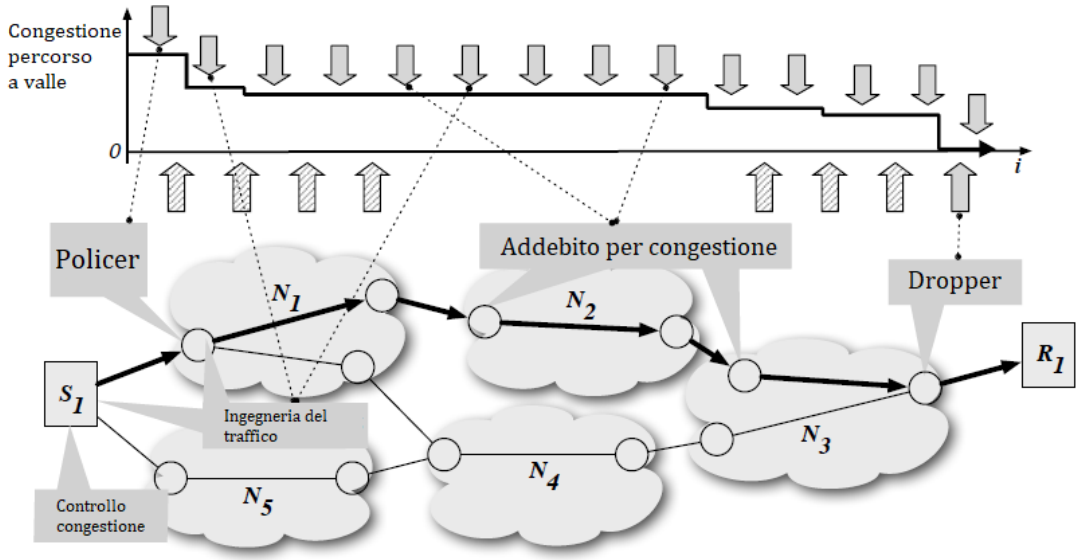

Figura 2.3. Scenario del re-feedback.

ciascun tipo di essi. Un pacchetto senza un feedback sulla congestione viene trattato come un pacchetto di richiesta e la larghezza di banda allocata per il traffico delle richieste è limitata al 5% della capacità del link in uscita. Ai pacchetti regolari viene data una priorità maggiore rispetto ai pacchetti legacy. Solo ai router di accesso e ai router considerati a rischio è richiesto di eseguire la generazione di feedback di congestione.

NetFence definisce tre tipi di feedback sulla congestione: nessun controllo, link sovraccarico e link sotto carico. I router di accesso collegati a un end-system inseriscono il feedback "nessun controllo" o "link sotto carico" nell'intestazione NetFence di ciascun pacchetto e questo feedback viene aggiornato a "link sovraccarico" lungo il percorso verso i destiniatari dal router che per primo si trova in congestione o che sta subendo un attacco. Il feedback sulla congestione viene quindi trasferito insieme ai pacchetti dati o ai pacchetti di ACK se viene utilizzato un protocollo come TCP, altrimenti viene rinviato in un pacchetto separato se vengono utilizzati protocolli di trasporto unidirezionali come UDP. Gli utenti devono inserire questo feedback nei loro pacchetti per evitare che questi vengano trattati come pacchetti legacy a bassa priorità. Un router di accesso mantiene un limitatore di velocità per ciascun mittente per limitare il normale traffico di pacchetti che attraversa un link a rischio congestione. Il router di accesso utilizza l'algoritmo di aumento additivo e diminuzione moltiplicativa per controllare il limite di velocità. Esso aumenta il limite di velocità in modo additivo o lo diminuisce in modo moltiplicativo a seconda che il feedback di congestione sia "link sotto carico" o "link sovraccarico".

NetFence utilizza funzioni hash crittografiche per garantire l'integrità del feedback sulla congestione. Per consentire ai router di accesso di convalidare il feedback sulla congestione generato dai router a rischio, NetFence richiede la condivisione di chiavi simmetriche tra AS attraverso uno scambio di chiavi pubbliche Diffie-Hellman incorporato nei messaggi di aggiornamento BGP.

Il vantaggio di NetFence rispetto ai precedenti meccanismi di controllo della

congestione è che fornisce un feedback non falsificabile sulla congestione. Uno dei suoi punti di forza è che riduce la necessità di verificare la capacità dei link (in questo caso, il feedback sulla congestione è la capacità del link) solo nei router di accesso e nei router a rischio, risparmiando così un significativo sovraccarico di elaborazione sui router della rete. Inoltre, NetFence riduce significativamente la quantità di informazioni di stato mantenute dai router implementando fair-queuing nei router di accesso.

Tuttavia, utilizzare questo metodo potrebbe non essere sufficiente per impedire la negazione del servizio a client legittimi, se il numero di client dannosi è molto elevato. NetFence deve essere combinato con meccanismi di rilevamento per identificare la maggior parte dei mittenti dannosi e impedire che il loro traffico entri nella rete. Un altro problema di NetFence è che aggiunge un'intestazione per il campo feedback di 28 byte per ogni pacchetto, il che è un overhead considerevolmente grande. Inoltre, la condivisione della chiave simmetrica tra ciascuna coppia di AS e la distribuzione di questa chiave a livello AS tra tutti i router di accesso all'interno di un AS è alquanto problematica considerando la configurazione aggiuntiva e il sovraccarico di gestione che esso richiede. Nel complesso, NetFence raggiunge un certo livello di miglioramento rispetto ai precedenti approcci di difesa DDoS basati sulla congestione e sulla capacità. Allo stesso tempo, integra però diversi meccanismi complessi e aumenta significativamente la complessità dei sistemi che distribuiscono NetFence.

2.3.2 Tolleranza ai guasti

I meccanismi di tolleranza ai guasti possono essere implementati a livello di hardware, software e di sistema. Nell'intento di mitigazione degli attacchi DoS, la tolleranza ai guasti viene spesso ottenuta replicando il servizio offerto su più nodi o aumentando le risorse utilizzate da quel servizio. Un esempio di un servizio che implementa una tecnica di replica di servizio per i sistemi di web hosting è XenoService, un'infrastruttura di rete distribuita di web host che risponde ad un attacco ad un sito web replicando il sito web rapidamente e ampiamente tra i server XenoService, consentendo così al sito attaccato di non perdere la propria disponibilità in rete. D'altra parte, la tecnica di aumento delle risorse, nota anche come capacità di over-provisioning, mira a tollerare gli attacchi DoS mediante un incremento nel numero delle risorse disponibili ad un servizio. I meccanismi di tolleranza ai guasti possono essere molto efficaci contro gli attacchi DoS, tuttavia di solito sono molto costosi da implementare e le risorse impiegate in aggiunta potrebbero essere sprecate poiché potrebbero non essere utilizzate durante il periodo di non attacco.

2.3.3 Resource Accounting

I meccanismi di resource accounting controllano l'accesso di ogni utente in base ai suoi privilegi e al suo comportamento. L'utente potrebbe essere un processo, una persona, un indirizzo IP o un insieme di indirizzi IP [18]. In questo modo, i meccanismi di resource accounting garantiscono il servizio agli utenti legittimi e non sospetti. Al fine di evitare il furto di identità dell'utente, di solito tali meccanismi

sono associati a meccanismi di accesso che verificano l'identità dell'utente. I meccanismi che testano la legittimità degli utenti e consentono l'accesso al server solo agli utenti legittimi appartengono a questa categoria.

2.4 Client puzzle come tecnica di mitigazione

Prima di introdurre i client puzzle, è necessario definire cosa sia un puzzle. Il concetto di puzzle fu proposto per la prima volta da Merkle in [17]. In questo articolo viene introdotto un sistema crittografico a chiave pubblica che utilizza i puzzle crittografici piuttosto che il sistema crittografico Diffie-Hellman.

In sostanza, un puzzle crittografico è definito da Merkle come un crittogramma che deve essere risolto. Nello schema di Merkle, il crittogramma è crittografato con una funzione crittografica e una parte della soluzione viene rivelata al risolutore. Pertanto, in questo schema, un puzzle è costituito da un testo in chiaro, un testo cifrato e una parte della chiave. I rimanenti pezzi sconosciuti della chiave sarebbero la soluzione al puzzle. Affinché il risolutore trovi la soluzione al puzzle, deve essere applicato un approccio brute-force che prova valori casuali per i bit rimanenti della chiave e quindi controlla il valore del testo cifrato per determinare se è stata trovata la chiave corretta.

Per applicare questa strategia a un sistema crittografico a chiave pubblica come fece Merkle, una persona (Marco) crea un gran numero di puzzle. Ogni soluzione del puzzle è composta da un ID e una chiave, entrambi variabili casuali. Quando una persona in particolare (Alice) desidera comunicare con Marco, Marco invia ad Alice un gran numero di puzzle creati in precedenza. Alice risolve un solo puzzle per recuperare l'ID e la chiave. Questa chiave è la chiave privata della comunicazione, memorizzata sia da Alice che da Marco. Dopo aver risolto un puzzle, Alice restituisce l'ID al server. Poiché Marco ha creato i puzzle in anticipo, ha una tabella di valori contenenti le coppie di ID e di chiave private. Pertanto, è possibile stabilire un accordo sulla chiave senza inviare il valore effettivo della chiave privata. Un attaccante interposto tra questi due utenti può conoscere solo il valore dell'ID e i puzzle inviati. Quindi, per trovare la chiave privata condivisa tra Alice e Marco, l'attaccante dovrebbe risolvere tutti i puzzle. Sebbene questo schema non sia ampiamente utilizzato nella pratica, esso è stato uno dei primi tentativi di introdurre un nuovo metodo di difesa basato sulla crittografia a chiave pubblica.

2.4.1 Client puzzle a livello rete

L'idea di base dei client puzzle è che ogni client risolva un puzzle prima di poter inviare traffico nella rete. Pertanto, i client puzzle sono progettati per limitare la velocità di generazione di pacchetti nei client che inviano grandi quantità di dati, in modo da aiutare la prevenzione di attacchi di DoS su larga scala.

Nei *congestion puzzle* di Wang e Reiter [21], ogni client è chiamato alla risoluzione di un certo numero di puzzle prima che i suoi pacchetti possano essere inoltrati a un router in congestione. Quando i client iniziano a inviare pacchetti verso un

particolare indirizzo IP, l'host destinazione invia continuamente pacchetti campione separati, insieme ai dati nei pacchetti normali che sta inviando. Quando un router di downstream rileva una congestione, ritrasmette i pacchetti campione modificando il numero del codice ICMP in modo che assomigli a un ping quando esso raggiunge la vittima (la vittima non deve così risolvere puzzle). Questo pacchetto verrà modificato per contenere le informazioni sul puzzle: un nonce e un livello di difficoltà. Quando il client riceve la sfida, inizia a risolvere continuamente puzzle e ad incorporare le soluzioni in pacchetti ICMP separati. Esso prende il nonce ricevuto dal router, crea il proprio nonce e utilizza questi due elementi per creare un puzzle basato su hash. Pertanto, il client non deve contattare il router congestionato per ottenere un nuovo puzzle. Le soluzioni vengono inviate in pacchetti ICMP inviati alla destinazione ma vengono intercettate dal router per la verifica del puzzle. Dopo la corretta verifica, per ogni puzzle risolto viene creato un token che viene aggiunto in un bucket di token del router congestionato. Quando arriva un pacchetto di dati, i token vengono rimossi dal bucket. Pertanto, questo funge da limitatore di velocità e consente ai client di inviare più pacchetti solamente se a sua volta essi risolvono più puzzle. Periodicamente, le informazioni sul puzzle (cioè il nonce e il livello di difficoltà) vengono aggiornate inviando la soluzione di un puzzle con le informazioni aggiornate. Pertanto, in questo schema, il doppio dei pacchetti viene inviato a un router congestionato. Ciò può creare un potenziale problema, perché un protocollo puzzle a livello IP dovrebbe limitare la quantità di traffico inviata alla vittima. È importante ricordare che uno degli obiettivi di un attacco DoS è quello di consumare la larghezza di banda vicino alla vittima. Un'altra considerazione in questo protocollo è che un utente malintenzionato può trarre vantaggio dal design del bucket di token inondando la rete di pacchetti (senza risolvere puzzle) e sperando che i client legittimi rimuovano i token inviando i loro pacchetti dati. Gli autori sono consapevoli di questo problema e lo chiamano problema del "free riding". Essi tentarono di risolvere questo problema introducendo un algoritmo di memorizzazione nella cache IP che alloca un bucket di token separato per quei client che inviano più dati rispetto ad altri o per quelli con un prefisso IP comune. Tuttavia, questo aggiunge molta complessità al loro schema e aumenta il sovraccarico sul sistema di memoria dei router.

Il protocollo presentato da Feng [20], chiamato Network Puzzles, richiede che i client e i router congestionati si scambino costantemente informazioni sui puzzle. A differenza dei congestion puzzle, ogni client può risolvere un puzzle solo quando gli sono state inviate le informazioni del puzzle dal router congestionato. Se un client dovesse risolvere un puzzle per ogni pacchetto inviato, avrebbe ripetutamente bisogno di richiedere le informazioni del puzzle al router congestionato prima di poter inviare i suoi pacchetti. Questo schema non consente una perfetta integrazione del protocollo di puzzle in IP perché non consente al client di creare i propri puzzle.

In tutti i protocolli di client puzzle si afferma che i puzzle debbano essere usati solamente durante un attacco in corso. Tuttavia, le prestazioni delle applicazioni di rete potrebbero risentire degli effetti collaterali della risoluzione dei puzzle in presenza di un attacco DoS. I ritardi aggiuntivi associati alla risoluzione dei puzzle potrebbero ostacolare alcune applicazioni, ma comunque ciò rappresenta un miglioramento delle prestazioni della rete durante un attacco DoS rispetto a quando i puzzle non vengono distribuiti. L'idea di implementare i client puzzle nel livello

rete e nel livello trasporto è ancora relativamente nuova nella comunità di ricerca, poiché ci sono stati pochissimi progetti concettuali di tale protocollo.

Client puzzle a livello trasporto

Negli ultimi anni, sono state proposte numerose varianti di schemi di client puzzle a livello trasporto. Juels e Brainard [13] hanno introdotto per la prima volta i client puzzle per prevenire attacchi DoS a livello trasporto. Inoltre, sono anche stati ideati client puzzle per aiutare a prevenire attacchi DoS ai protocolli di autenticazione [5].

In [22], Wang e Reiter presentano un protocollo di client puzzle chiamato *puzzle auction* (letteralmente "puzzle ad asta") che è stato implementato e incorporato nello stack TCP in Linux. I puzzle auction sono stati progettati per consentire ai client, con un kernel modificato, di fare "un'offerta" per una connessione. Questa offerta rappresenta la difficoltà del puzzle e un'offerta più alta implica un puzzle più difficile da risolvere da parte dell'offerente. Nel loro schema, il client fa un'offerta per una connessione risolvendo un puzzle a una certa difficoltà e può ripresentare un'offerta per quella connessione risolvendo un altro puzzle di un livello di difficoltà superiore e quindi ritrasmettere la richiesta. Alla fine della ricezione, il server concede la connessione a coloro che hanno fatto l'offerta più alta o a coloro che hanno risolto il puzzle più difficile. Consentire a un client di impostare il livello di difficoltà può consentire a un attaccante, in controllo di alcuni zombie, di aumentare intenzionalmente il livello di difficoltà e di superare le offerte degli altri client. Gli autori affermano che la maggior parte dei programmi che vengono eseguiti sugli host zombie sono progettati per funzionare in modo silente per far sì che gli utenti non si accorgano della loro presenza. Gli autori hanno così pensato che risolvere i puzzle ripetutamente e di difficoltà incrementale segnalerà a un utente che il loro computer è stato compromesso in quanto saranno sottoposti a un carico computazionale sempre più grande. Questo non è sempre vero, soprattutto se l'attacco viene lanciato durante periodi di inattività dell'host o se l'host non utilizza pesantemente il sistema. Per i motivi sopra indicati, il livello di difficoltà di uno schema puzzle dovrebbe sempre essere controllato dal server o dall'host finale che distribuisce i puzzle.

Per compatibilità con le versioni precedenti, lo schema degli auction puzzle ha un metodo per consentire a un client con un kernel non modificato di completare una connessione durante un attacco. Sfortunatamente, questa implementazione presenta potenziali vulnerabilità. Nell'implementazione, un client con un kernel non modificato non risolve un puzzle; invece, quando il server riceve una richiesta di connessione, è il server che calcola l'hash di un nonce, indirizzo IP di origine, porta di origine, indirizzo IP di destinazione, porta di destinazione, numero di sequenza iniziale e un altro valore casuale. Se l'output della funzione hash soddisfa il livello di difficoltà, la connessione è completata. Un client può stabilire una connessione solo effettuando ripetutamente tentativi e sperando che il server possa concederla. Questo schema viene chiamato "bid and query". Tale soluzione viola chiaramente la prima caratteristica di un protocollo di client puzzle menzionato antecedentemente, perché il client e il server sono entrambi tenuti a eseguire una quantità di lavoro simile per completare la connessione. Questa vulnerabilità può essere sfruttata da un client dannoso.

Waters [23] propone un nuovo metodo per costruire i puzzle client a livello trasporto e descrive brevemente come la loro implementazione potrebbe essere modificata per i puzzle a livello rete. Nel suo schema, afferma che la maggior parte dei client non dovrebbe attendere la risoluzione di un puzzle per completare una richiesta di connessione. Per questo motivo, introduce un metodo complesso per risolvere i puzzle prima che le richieste di connessione vengano effettivamente accettate, eliminando così il tempo che un client trascorre in attesa che un puzzle venga risolto. In questo caso, i puzzle vengono risolti in un certo periodo di tempo e le risposte del puzzle vengono utilizzate in un periodo di tempo successivo, utilizzando quindi soluzioni calcolate precedentemente. Il problema è che quando un client si connette per la prima volta al server, deve attendere l'inizio del periodo di tempo successivo per poter utilizzare le risposte calcolate antecedentemente. L'autore suggerisce che questo periodo di tempo sia dell'ordine dei minuti e che nei loro esperimenti arrivano ad utilizzare anche periodi di 20 minuti. Ciò significa che un client, che si era appena connesso di recente, avrebbe dovuto attendere circa 20 minuti prima di stabilire una connessione con un server remoto. Nel suo articolo, uno dei principali punti di forza della sua soluzione è la riduzione del tempo di verifica del puzzle. Poiché i puzzle vengono risolti in un periodo di tempo specifico, le soluzioni del puzzle vengono memorizzate per quel periodo di tempo e quello successivo. Durante il periodo di tempo successivo, la verifica di un puzzle può essere eseguita semplicemente con una ricerca in una tabella di stato. In termini di tempo di verifica, questo è il metodo più veloce per verificare un puzzle. Tuttavia, lo spazio di archiviazione e la complessità vengono sacrificati a scapito del tempo di verifica del puzzle.

2.4.2 Client puzzle a livello applicazione

Sono stati proposti anche client puzzle per contrastare gli attacchi che possono verificarsi a livello applicazione. In [8], gli autori hanno modificato un server web per supportare i client puzzle. In TLS, può verificarsi un attacco DoS quando viene stabilita una connessione sicura, in quanto il server deve eseguire decrittazioni RSA che richiedono un uso intensivo della CPU. Se utenti malintenzionati costringono ripetutamente il server a eseguire queste operazioni, la CPU del server sarà totalmente utilizzata e il server non sarà più in grado di offrire servizio agli altri client. Gli autori di questo articolo propongono uno schema di client puzzle, in cui il client risolve un puzzle prima che il server esegua qualsiasi altro compito. Pertanto, il protocollo di puzzle è in grado di trasferire il costo della creazione della connessione sul client, piuttosto che sul server. Questo protocollo è specifico dell'applicazione ed è stato progettato per correggere un exploit che ha consentito a un utente malintenzionato di esaurire le risorse di un server TLS. Utilizzando un protocollo di client puzzle, è stato più difficile per l'attaccante esaurire con successo le risorse.

Uno degli usi più promettenti dei client puzzle a livello di applicazione è la loro distribuzione per combattere la posta indesiderata o lo spam. Gli autori di [10] sono stati i primi a proporre uno schema in cui un utente risolve un puzzle prima che venga inviata un'e-mail. Poiché gli spammer, in media, inviano più posta rispetto agli utenti di posta normali, gli autori sostengono che forzando ogni mittente di posta elettronica a risolvere un puzzle crittografico per ogni messaggio di

posta elettronica, si limiterebbero gli effetti causati da utenti malintenzionati. Nel loro lavoro futuro, hanno anche proposto di utilizzare puzzle basati sulla memoria piuttosto che puzzle basati su calcoli [9]. I puzzle basati sulla memoria erano originariamente presentati in [3]. Nel loro articolo, la loro idea era di sviluppare un puzzle migliorato che costringa i client a dedicare le loro risorse di memoria per risolvere problemi moderatamente difficili. Nonostante ciò, sono stati proposti pochissimi protocolli di client puzzle che utilizzano questo tipo di soluzione poiché sono difficili da creare.

Oltre a ricercare un protocollo proof-of-work, come i client puzzle, all'interno del livello di trasporto, Back [6] ha considerato le possibilità di utilizzare tale protocollo all'interno di varie applicazioni. L'autore ha proposto uno schema chiamato *Hashcash*, che può essere distribuito all'interno di un'applicazione per limitare lo spam e le richieste di servizio in un file system.

Capitolo 3

Analisi del sistema e threat model

Lo scopo di questo capitolo è presentare l'analisi di un sistema client-server reale che utilizza un meccanismo di difesa basata su puzzle. Inoltre, verrà fornito il threat model che verrà tenute in conto in fase di sperimentazione e da cui il sistema proposto dovrà difendersi.

3.1 Modello del sistema

Il sistema considerato è un'architettura client-server. Un *server* è un processo che fornisce contenuto o servizi a un gran numero di client; un *client* è un processo che richiede un servizio da un server. I termini client e server vengono anche utilizzati per indicare le macchine che eseguono i processi rispettivamente nel client e nel server. I client sono ulteriormente classificati come client legittimi, che non hanno alcun obiettivo dannoso, e client malevoli, che invece hanno intenti dannosi verso il server. Un utente malintenzionato è un utente che controlla i client dannosi. Nel contesto degli attacchi DoS, un utente malintenzionato tenta di sopraffare il server con richieste illegittime al fine di negare o interrompere il normale servizio ai clienti legittimi.

Il termine *transazione* si riferisce a una sequenza di scambi di messaggi tra il client e il server che comporta la concessione o il rifiuto di una singola richiesta di servizio. Il server può richiedere al client di risolvere un puzzle, che può comportare l'interazione con il server o altri proxy server più volte, ma l'intero scambio di messaggi riferiti ad uno stesso puzzle è ancora da considerarsi come parte di una singola transazione. Una singola transazione di solito ha sei messaggi: una richiesta iniziale di accesso al server I_1, una risposta iniziale del server R_1 che può includere la richiesta di risoluzione di un puzzle, un'ulteriore richiesta del client I_2 che contiene la soluzione del puzzle, una risposta di accettazione/rifiuto del server R_2, una richiesta della risorsa richiesta del client I_3 e una risposta finale del server R_3 che contiene la risorsa cercata. Quando il meccanismo di puzzle non è attivo, la transazione include solo i messaggi I_1, R_1, I_3 e R_3. Quando il protocollo puzzle prevede che il client interagisca con uno o più proxy server, l'insieme dei messaggi incluso in una singola transazione assume la forma di I_1, R_1, IP_1, RP_1, ..., IP_k, RP_k, I_2, R_2, I_3 e R_3, dove IP_1, RP_1, ..., IP_k, RP_k sono messaggi scambiati tra il client e i proxy server.

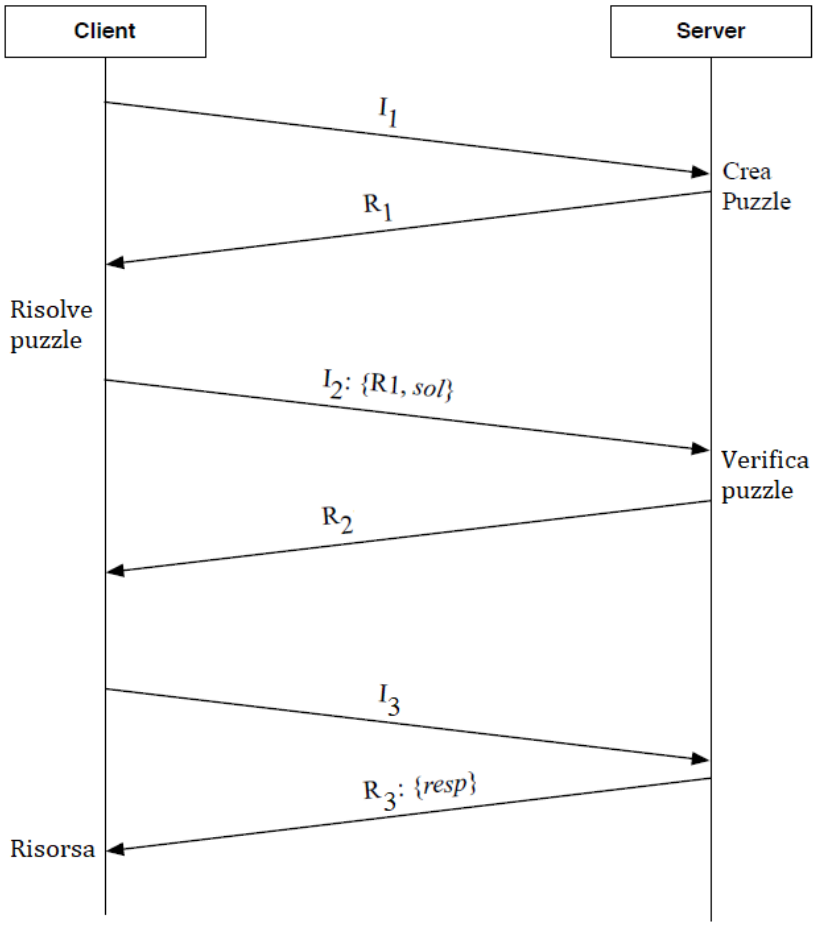

Figura 3.1. Una transazione client-server in un protocollo di puzzle

3.1.1 Analisi matematica

Di seguito segue un'analisi matematica di un protocollo di puzzle generale che permette di identificare quali siano i parametri da adottare per poter permettere una difesa efficace.

Client legittimi

Per semplificare l'analisi, si assume che ogni client legittimo possa elaborare f_l istruzioni al secondo (i/s) e debba eseguire c istruzioni per completare ogni transazione (i/tr), quindi ogni cliente può raggiungere transazioni f_l/c al secondo (tr/s). Ciò non significa che ogni client invierà sicuramente le sue richieste a questo rate di invio, ma il rate effettivo potrebbe essere anche più basso. Quando è attivo il protocollo di puzzle, un client è tenuto a risolvere un puzzle che richiede in media τ istruzioni, quindi il tasso medio di transazione per secondo di un singolo client è dato da:

$$\frac{f_l}{c+\tau} \tag{3.1}$$

Nella precedente equazione, τ viene chiamato *difficoltà del puzzle*.

Quando è attivo un protocollo di delay-puzzle (in cui la risoluzione viene ritardata di un tempo δ), ogni puzzle richiede in media δ secondi per la risoluzione, quindi ogni transazione richiede $\delta + \frac{c}{f_l}$ secondi e il tasso medio di transazioni per secondo diventa:

$$\frac{1}{\delta + \frac{c}{f_l}} = \frac{f_l}{c + \delta f_l} \tag{3.2}$$

Supponendo che il numero di client legittimi attivi nel sistema sia N_l, il carico massimo che può essere generato da tutti i client legittimi è:

$$L_l = N_l \frac{f_l}{c + \tau} \tag{3.3}$$

e per i delay-puzzle è:

$$\hat{L}_l = N_l \frac{f_l}{c + \delta f_l} \tag{3.4}$$

Client malevoli

I client malevoli possono essere modellati in modo simile, con le differenze della potenza di elaborazione delle macchine client e del loro numero. Supponendo che il numero di client malevoli attivi nel sistema sia N_m e che ogni client malevolo possa elaborare f_m istruzioni al secondo, il carico massimo che può essere generato da tutti i client malevoli è:

$$L_m = N_m \frac{f_m}{c + \tau} \tag{3.5}$$

e per delay puzzle è:

$$\hat{L_m} = N_m \frac{f_m}{c + \delta f_m} \tag{3.6}$$

Server

Il server può elaborare F istruzioni al secondo e ogni richiesta richiede in media C istruzioni per l'elaborazione da parte del server, quindi il server ha una capacità $\mu = \frac{F}{C}$ transazioni al secondo. In base alle caratteristiche della coda del server, il fattore di utilizzo del server vale $\rho = \frac{\lambda}{\mu}$, dove λ è il tasso di arrivo delle richieste. L'obiettivo è utilizzare il protocollo di puzzle per tenere sotto controllo l'utilizzo del server e per farlo bisogna agire sul parametro λ. In questo caso, il tasso di arrivo di richieste che si può controllare si riferisce solo al tasso di arrivo delle richieste accompagnate da una soluzione puzzle valida, poiché non è possibile controllare la velocità con cui i client possano inviare richieste non valide.

Supponendo che ρ sia l'utilizzo attuale del server e ρ^* sia l'utilizzo che si vuole tentare di ottenere, essi possono essere scritti come:

$$\rho = \frac{\lambda}{\mu} \tag{3.7}$$

e

$$\rho^* = \frac{\lambda^*}{\mu} \tag{3.8}$$

Combinando le equazioni, si ottiene:

$$\lambda^* = \frac{\lambda \rho^*}{\rho} \tag{3.9}$$

Nella precedente espressione, λ^* è il valore di quanto dovrebbe valere il rate di arrivo se si vuole che l'utilizzo del server sia pari a ρ^* e che la somma del carico offerto dai client legittimi e malevoli sia uguale a λ^*. Perciò, λ^* può essere anche scritto come:

$$\lambda^* = L_l + L_m = N_l \frac{f_l}{c + \tau} + N_m \frac{f_m}{c + \tau} \tag{3.10}$$

e per i delay-puzzle come:

$$\lambda^* = \hat{L_l} + \hat{L_m} = N_l \frac{f_l}{c + \delta f_l} + N_m \frac{f_m}{c + \delta f_m} \tag{3.11}$$

Poiché non si conoscono le frequenze medie delle CPU dei client, sia legittimi sia malevoli, si può semplicemente utilizzare una frequenza CPU media complessiva f per sostituirle. Pertanto, l'equazione 3.10 diventa:

$$\lambda^* = L_l + L_m = N_l \frac{f}{c + \tau} + N_m \frac{f}{c + \tau} = (N_l + N_m) \frac{f}{c + \tau} \tag{3.12}$$

La somma $(N_l + N_m)$ può essere sostituita con N, che è il numero totale di client attualmente attivi nel sistema e si impone $c = 0$, essendo $c << \tau$. Successivamente, l'equazione 3.12 diventa:

$$\lambda^* = N \frac{f}{\tau} \tag{3.13}$$

Risolvendo l'equazione 3.13 per la difficoltà media del puzzle τ, si ottiene:

$$\tau = \frac{N f}{\lambda^*} \tag{3.14}$$

Come ci si poteva aspettare, la difficoltà media del puzzle dovrebbe essere determinata collettivamente dal numero totale di client attivi, dalla potenza di elaborazione media dei client e dal tasso di arrivo delle richieste desiderato. Dato che N non è una costante e varia nel tempo, si può riscrivere l'equazione 3.14 come segue per evidenziare la dinamicità di N:

$$\tau(t) = \frac{N(t) f}{\lambda^*} \tag{3.15}$$

La formula per la difficoltà del puzzle 3.15 ci dice quanta capacità computazionale per transazione dovrebbe impiegare in media ogni client, ma non dice quale

sia la difficoltà del puzzle per ogni singolo client. Per determinare la difficoltà del puzzle per ogni singolo client, è necessario anche tenere conto del contributo che ciascun client ha sulla capacità totale offerta dal server.

Allo stesso modo, la difficoltà δ nei delay puzzle può essere derivata dall'equazione 3.11 come segue:

$$\delta(t) = \frac{N(t)}{\lambda^*} \tag{3.16}$$

3.2 Threat model

Si assume che le risorse di rete siano sufficientemente grandi da gestire tutto il traffico di rete e che l'obiettivo degli attacchi sia la capacità computazionale del server. In particolare, i client malevoli sono bot o macchine zombie che sono controllati da un attaccante. L'attaccante può intercettare tutti i messaggi inviati tra un server e un qualsiasi client legittimo. Inoltre, l'utente malintenzionato può modificare solo un numero limitato di messaggi client inviati al server. Questa assunzione è da considerarsi ragionevole in quanto se un utente malintenzionato potesse modificare tutti i messaggi dei client, potrebbe semplicemente lanciare un attacco eliminando tutti i messaggi inviati da altri client.

Per effettuare un attacco, l'attaccante può utilizzare una o una combinazione delle seguenti strategie:

- **Contraffazione:** Un utente malintenzionato può inviare soluzioni puzzle non valide;
- **Time shifting:** L'attaccante può raccogliere un gran numero di soluzioni dei puzzle prima dell'attacco e usarle per inviare un gran numero di richieste durante l'attacco;
- **Collusione:** L'utente malintenzionato può condividere soluzioni puzzle tra client malevoli per ridurre il costo computazionale complessivo per la risoluzione del puzzle;
- **Replay:** L'attaccante può inviare più volte una stessa soluzione di un puzzle valida;
- **Spoofing:** L'attaccante può usare indirizzi sorgente falsificati durante un attacco.
- **Risoluzione di puzzle simultanei:** L'attaccante può risolvere più puzzle contemporaneamente in modo da ridurre il tempo totale per risolverli tutti.

Gli attacchi che utilizzano una di queste strategie possono essere indicati come *attacco di contraffazione*, *attacco time-shifting*, *attacco di collusione* (o *attacco cookie jar*), *attacco replay*, *attacco spoofing* e *attacco simultaneo di risoluzione puzzle*. Un attaccante può anche lanciare altri attacchi al protocollo di puzzle, intervenendo anche nelle fasi di costruzione del puzzle, la sua distribuzione o la sua verifica.

In questa tesi gli attacchi al protocollo sono definiti collettivamente come *attacchi resistenti ai puzzle*. Nello sviluppo della soluzione proposta sono stati usati alcuni algoritmi crittografici, ma non sono stati presi in considerazione attacchi brute force o di cripto-analisi contro gli algoritmi stessi. Tuttavia, è necessario che gli algoritmi di costruzione, distribuzione e verifica dei puzzle siano sicuri in modo tale che non diventino vulnerabili ad *attacchi di complessità algoritmica*[1].

[1]Un attacco di complessità algoritmica è una forma di attacco informatico che sfrutta casi noti in cui un algoritmo mostra le sue debolezze. Questo tipo di attacco può essere utilizzato per ottenere negazione di servizio.

Capitolo 4

Progettazione

I protocolli di puzzle sono stati proposti per difendersi dagli attacchi DoS allo scopo di bilanciare il carico computazionale del server rispetto al carico computazionale dei client. La risoluzione di un puzzle in genere richiede l'esecuzione di un gran numero di operazioni crittografiche, come hashing, moltiplicazioni modulari, ecc. Pertanto, il numero di richieste che il server può soddisfare ai client è limitato dalle proprie risorse di calcolo disponibili.

Sebbene i metodi di mitigazione DoS basati su puzzle siano promettenti, la maggior parte delle soluzioni esistenti si concentrano sulla costruzione di puzzle, sulla verifica e sull'analisi della sicurezza del protocollo, ma pochi discutono degli aspetti pratici di questo tipo di difesa. I meccanismi di difesa basati su puzzle non sono ampiamente diffusi nella pratica, perché al momento non è presente un modello di design chiaramente condiviso. Principalmente, in letteratura ci sono tre problemi fondamentali esistenti che non sono adeguatamente affrontati:

1. non esiste un metodo chiaro per determinare dinamicamente la difficoltà dei puzzle che tenga conto del costo computazionale previsto di una richiesta ricevuta, in particolare da quali parametri essa debba dipendere;

2. si presume che il protocollo di puzzle debba essere attivato quando il server è sotto attacco, ma non è chiaro come determinare quando il protocollo debba essere attivato o no;

3. poche pubblicazioni esistenti sulla difesa DoS basata su puzzle menzionano l'attacco replay introdotto nel capitolo precedente e nessuna propone una soluzione funzionante per prevenirlo.

Lo scopo di questo capitolo è introdurre una soluzione ai problemi precedenti. Successivamente, verrà presentato lo sviluppo di un nuovo protocollo di puzzle, costruito sulla base di tali considerazioni.

4.1 Difficoltà del puzzle

Il concetto di difficoltà del puzzle è menzionato in quasi tutta la letteratura sui puzzle, tuttavia nessuno fornisce un modello matematico utilizzabile o una formula per calcolarlo. Dean e Stubblefield [8] hanno suggerito di utilizzare un valore empirico di 20 bit per puzzle basati su hash, che essi adottarono per proteggersi dagli attacchi DoS contro il protocollo SSL. Wang [22] e Parno [19] hanno proposto di utilizzare una determinazione "ad asta" per la difficoltà dei puzzle, in cui i client e non il server determinano la difficoltà dei puzzle che sono tenuti a risolvere, per aumentare le loro possibilità di ottenere il servizio richiesto. Utilizzando tali meccanismi, un client deve fare molti tentativi prima di trovare una difficoltà di puzzle che possa consentire loro di accedere al servizio; alcuni sottoinsiemi di client potrebbero non essere in grado di acquisire servizio nei casi in cui gli attaccanti computazionalmente forti possono elevare la difficoltà del puzzle a livelli che suddetti client non possono risolvere.

Il protocollo backbone BitCoin utilizza un meccanismo di puzzle basato su hash e afferma l'importanza di determinare la difficoltà del puzzle [11], ma non fornisce come calcolarla se non suggerendo di tenere conto del numero di utenti nel sistema.

Laurie e Clayton [15] mostrano che i puzzle crittografici o i sistemi proof-of-work non funzionano se la difficoltà del puzzle è troppo alta. Groza e Warinschi [12] sottolineano la mancanza di un trattamento rigoroso dei parametri (come la difficoltà del puzzle) dei protocolli di difesa DoS basate su puzzle e sottolinea il fatto che la maggior parte delle proposte si basano su supposizioni empiriche, ad es., quando un server è sotto attacco la durezza del puzzle viene aumentata su osservazioni empiriche; ad esempio, la migliore protezione viene raggiunta quando la difficoltà è impostata su una certa soglia. Essi forniscono un limite al livello massimo di difficoltà del puzzle e dimostrano che non esiste alcun vantaggio di protezione DoS nel impostare la difficoltà del puzzle al di sopra di tale soglia.

In questa tesi, per determinare quale difficoltà debba essere assegnata a un determinato client, verrà sfruttati l'analisi effettuata nel capitolo 3. In particolare, si è visto che la difficoltà media del puzzle debba dipendere dal numero di client attivi nel sistema e dal tasso di arrivo delle richieste desiderato. Per fare ciò, verranno inserite strutture dati in grado di monitorare il numero di client attivi e verrà utilizzato un metodo di attivazione del protocollo basato su due soglie, come descritto in seguito.

4.2 Determinazione dell'attivazione del protocollo

Determinare quando attivare o disattivare correttamente i puzzle è importante, perché consente alla difesa DoS di rispondere rapidamente quando il server è sotto attacco e di eliminare l'onere della risoluzione dei puzzle sui client quando non vi è alcun attacco.

La maggior parte della letteratura esistente sui puzzle non fornisce informazioni su come determinare i tempi di accensione/spegnimento dei puzzle o suggerisce di

attivarli quando il server è sotto attacco o quando il carico del server è al di sopra di una certa soglia. Dean e Stubblefield [8] forniscono la discussione più dettagliata sull'argomento e propongono di usare due soglie separate, una per attivare i puzzle e una per disattivarli. Essi, tuttavia, forniscono solo numeri di soglia specifici adatti in circostanze molto limitate per l'handshake TLS in cui cercavano di proteggersi.

Durante lo sviluppo, si utilizza un approccio euristico per determinare il punto di attivazione dei puzzle. Si usano due livelli di soglia: una soglia di attivazione λ_{on} e una soglia di disattivazione λ_{off}, ma ci sono differenze importanti. Innanzitutto, si confrontano i tassi di arrivo delle richieste λ con queste due soglie, oltre a verificare il carico del server. In un contesto DoS, il rate di arrivo delle richieste può raggiungere il suo picco molto prima che il server capisca che il suo utilizzo abbia oltrepassato la soglia preimpostata, per questo è utile anche osservare il tasso delle richieste in arrivo. In secondo luogo, si imposta la soglia di attivazione del puzzle λ_{on} al tasso di arrivo medio previsto e si imposta la soglia di disattivazione λ_{off} pari alla metà di λ_{on}, ovvero, $\lambda_{off} = \frac{\lambda_{on}}{2}$

4.3 Soluzione per prevenire gli attacchi replay

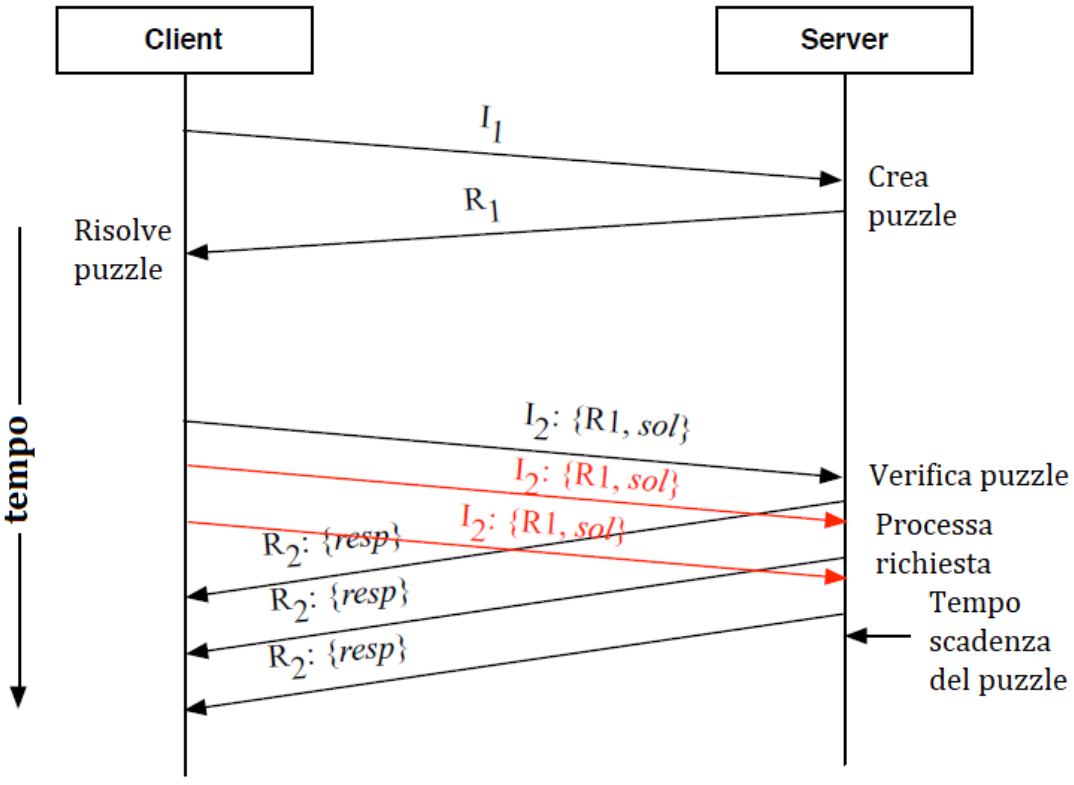

Figura 4.1. Un esempio di attacco replay

In un protocollo di puzzle, un client malintenzionato può tentare di amplificare l'impatto del suo attacco utilizzando ripetutamente la stessa soluzione di uno stesso puzzle. In particolare, il terzo messaggio I_2, come descritto nel protocollo nella Sezione 3.1, che include la soluzione del puzzle, può essere re-inviato da un client

malevolo al server fino alla scadenza del puzzle corrispondente, come dimostrato in Figura 4.1.

Aura [5] suggerisce di verificare se una soluzione sia replicata o meno, ma non fornisce informazioni specifiche sul modo in cui ciò dovrebbe essere verificato. Wang e Reiter [23] propongono di verificare il nonce del client che invia una richiesta appena ricevuta tra tutte le richieste esistenti nella coda delle richieste per verificare se esista già una richiesta con tale nonce e quindi annullare tale richiesta se il nonce è già presente. Feng [20] suggerisce di associare il puzzle a un flusso o a un pacchetto specifico per richiedere la risoluzione di un nuovo puzzle per ogni nuovo flusso o nuovo pacchetto. Lakshminarayanan [14] ha suggerito di mantenere ogni puzzle sul server fino a quando non venga ricevuta una risposta dal client o non è trascorso un determinato periodo di tempo dalla sua creazione.

In un'applicazione tipica di un protocollo crittografico, gli attacchi replay possono essere prevenuti stabilendo un ID di sessione univoco per ogni esecuzione del protocollo e utilizzando nonces lato client e server. Il server, tuttavia, non può mantenere questi identificatori univoci per sempre, quindi è necessario combinare il mantenimento dello stato con il tempo di scadenza del puzzle per ottenere una protezione efficiente ed efficace contro gli attacchi replay.

Una soluzione per mantenere lo stato limitato nel tempo consiste nell'utilizzare una tabella di hash per mantenere la soluzione puzzle recentemente inviata, in modo tale da poter trovare una soluzione puzzle già utilizzata cercandola in questa tabella hash. Quando la soluzione viene inserita nella tabella, essa viene salvata con il timestamp di arrivo, in modo che possa esserne verificata facilmente la sua validità. Idealmente, una soluzione puzzle dovrebbe essere rimossa dalla tabella hash subito dopo la scadenza del puzzle corrispondente, ovvero $now() > t_{exp}$, dove t_{exp} indica il tempo di scadenza del puzzle.

4.4 Progetto

Il progetto integra il protocollo di puzzle, la difesa dagli attacchi replay e l'algoritmo di attivazione/disattivazione dei puzzle che sono stati introdotti nelle sezioni precedenti. Il meccanismo adottato segue la costruzione dei puzzle basata su hash comunemente usata e il protocollo è sicuro rispetto al threat model definito nella Sezione 3.2. Il codice sorgente del server è scritto adottando il linguaggio Java e le misure di sicurezza sono ottenute utilizzando la libreria Java JCE (Java Cryptographic Extension).

Prima di cominciare la descrizione dell'implementazione, bisogna considerare le seguenti assunzioni fatte prima dello sviluppo del progetto:

- Gli aggressori sono generalmente più aggressivi e inviano più richieste rispetto ai client legittimi.
- Il server durante un attacco può elaborare ogni pacchetto in arrivo e inviare risposte a ciascun client.

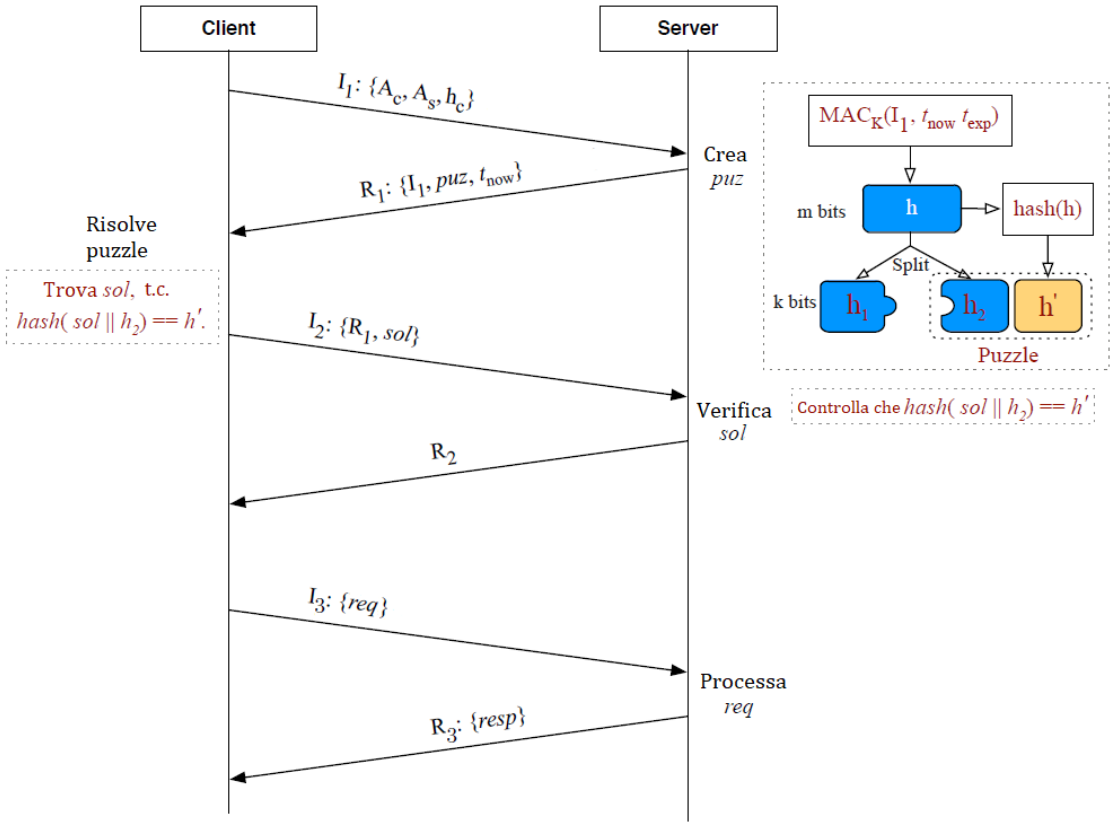

Figura 4.2. Interazione client-server con la risoluzione di un puzzle

Il protocollo di comunicazione client-server è costituito da sei scambi di messaggi, inviati tramite il protocollo di trasporto UDP:

1. richiesta iniziale di accesso del client I_1;
2. risposta iniziale R_1 del server che include un puzzle da risolvere;
3. richiesta di accesso I_2 del client che contiene la soluzione puzzle;
4. risposta di autorizzazione all'accesso R_2 del server;
5. richiesta I_3 della risorsa dal client;
6. risposta R_3 del server.

La figura 4.2 illustra l'interazione del client-server nel protocollo.

Poiché la soluzione proposta si basa sul protocollo di trasporto UDP, è necessaria una tabella di stato per tenere traccia dei vari client che desiderano accedere al servizio. Per consentire ciò, le informazioni sui client vengono salvate in una tabella di hash nella memoria del server e i client vengono differenziati utilizzando l'indirizzo IP di origine e la porta remota da cui sono stati inviati i pacchetti. Ciò si rivela utile anche per prevenire gli attacchi replay, in quanto il metodo proposto nella sezione precedente suggeriva di usare una tabella di stato dove mantenere il

tempo di scadenza del puzzle. In questo modo, è sufficiente solamente aggiungere un campo nelle entry della struttura dati salvata sul server per poter prevenire questo tipo di attacco.

Il server è single-threaded e analizza ogni pacchetto ricevuto, in base allo stato del suo mittente. Tutte le funzionalità che richiedono l'uso del puzzle sono offerte dalla classe `Puzzle`, che gestisce tutte le operazioni relative all'invio, alla ricezione e alla costruzione di puzzle.

Prima di tutto, il client comincia la comunicazione con un messaggio di richiesta iniziale I_1 che contiene le seguenti informazioni:

$$I_1 = \{A_c, A_s, req, h_c\},$$

dove A_c e A_s sono degli identificatori univoci del client e del server rispettivamente, come il loro indirizzo IP o il loro URI (Uniform Resource Identifier); req è la risorsa richiesta dal client, in questa tesi l'identificativo di un oggetto salvato in un database; h_c si ottiene calcolando l'HMAC (Hash-based Message Authentication Code) dei tre precedenti elementi ed è così ottenuto:

$$h_c = hmac(K_c, A_c \| A_s \| req)$$

dove K_c è una chiave segreta utilizzata dal client per crittografare le proprie richieste che utilizza l'algoritmo SHA-256 e $\|$ è l'operatore di concatenazione tra stringhe. h_c mantiene l'integrità del messaggio del client e svolge anche il ruolo di *client nonce*. Il codice che implementa tali istruzioni è mostrato di seguito.

```
private void sendRequest() {
       try {
              KeyGenerator keygen =
                 KeyGenerator.getInstance("Hmac256");
              keygen.init(128);
              SecretKeySpec privKey = new
                 SecretKeySpec(keygen.generateKey().getEncoded(),
                 "Hmac256");
              StringBuffer messageForHash = new StringBuffer();
              StringBuffer message = new StringBuffer();
              String Ac =
                 Inet4Address.getLocalHost().getHostAddress().toString();
              byte[] hmacClient, messageSend;

              messageForHash.append(Ac).append("127.0.0.1").append("127.0.0.1
                 //Ac||As||req
              hmacClient = Functions.calcHmacSha256(privKey,
                 messageForHash.toString().getBytes("UTF-8"));
                 //request's hmac
              message.append(Ac).append(",").append("127.0.0.1").append(",").
                     .append(Functions.bytesToHex(hmacClient));

              messageSend = message.toString().getBytes();
```

```
                DatagramPacket dp = new
                    DatagramPacket(messageSend,messageSend.length,
                                serverAddress, serverPort);
                ds.send(dp);
        }catch(Exception e) {
                e.printStackTrace();
        }
}
```

Alla ricezione di un messaggio, il server verifica se la coppia indirizzo IP di origine/porta remota del client sia già presente nella sua tabella di stato; se ciò non viene verificato, significa che il pacchetto ricevuto è il primo pacchetto della comunicazione di protocollo, quindi il server aggiunge il client nella tabella con il numero di richieste ricevute da quel specifico client impostato a 0, altrimenti aumenta il numero di richieste ricevute da quel client di 1, quindi calcola il numero di bit necessari per costruire il puzzle in base al numero di richieste di arrivo da quel client e il rate di arrivo delle richieste desiderato, come proposto nel capitolo 3. Il metodo che fornisce questa funzione è il metodo `generatePuzzle()`, che riceve un vettore di byte e la dimensione del puzzle come input e salva il puzzle nella tabella di stato, insieme al suo timestamp t_{now}. Per determinare la dimensione, il server utilizza il numero di accessi provenienti dalla stessa origine e la soglia di accessi consentiti, che sono impostati all'inizio del programma. Se il numero di accessi è inferiore alla soglia indicata, la difficoltà del puzzle è impostata su -1 e nessun puzzle viene proposto. Altrimenti si crea un puzzle *puz* come segue (il server utilizza l'algoritmo introdotto nella Sezione 4.2 per determinare l'attivazione/disattivazione del protocollo):

$$puz = \{h_2, h'\},$$

dove, $h_2 = [h]_{m-k}$ rappresenta i $m - k$ bit più bassi di $h, h' = hash(h)$ e h è un hash digest di m bit calcolato come segue:

$$h = hmac(K_{puz}, (I_1, t_{now})),$$

dove, K_{puz} è la chiave segreta del server utilizzata unicamente per creare i puzzle e t_{now} è il tempo il cui viene generato il puzzle. La creazione del puzzle *puz* è illustrata in Figura 4.2. A seguire è riportato il codice che genera tale puzzle.

```
public void generatePuzzle(byte[] data, int nBitPuzzle) {
        try {
                String message;
                MessageDigest messageDigest =
                    MessageDigest.getInstance("SHA-256");
                timestamp = new
                    Timestamp(System.currentTimeMillis());
                BigInteger h2Big;
                this.nBitPuzzle = nBitPuzzle;
                if(nBitPuzzle > 0) {
                        message = new String(data,
                            StandardCharsets.UTF_8);
```

```
                        //System.out.println(new
                           StringBuffer(message).append(timestamp));
                        h = Functions.calcHmacSha256(puzzleKey, new
                           StringBuffer(message).append(timestamp)
                                      .toString().getBytes("UTF-8"));
                        hp = messageDigest.digest(h); //h'
                        h2Big = new BigInteger(h.clone());
                        for(int i = 0; i < nBitPuzzle; i++)
                               h2Big = h2Big.clearBit(i);
                        h2 = h2Big.toByteArray();
                }
        }catch(Exception e) {
                e.printStackTrace();
        }
}
```

Per salvare i valori `h`, `hp`, `h2` da 32 bit viene usata la classe Java `BigInteger`, che permette di rappresentare un valore di qualsiasi lunghezza in complemento a due.

Il codice che genera le chiavi del puzzle è mostrato qui sotto.

```
public static void generatePuzzleKey() {
        try {
                KeyGenerator keygen =
                   KeyGenerator.getInstance("Hmac256");
                keygen.init(128);
                byte[] key = keygen.generateKey().getEncoded();
                puzzleKey = new SecretKeySpec(key, "Hmac256");
        }catch(Exception e) {
                e.printStackTrace();
        }
}
```

Il codice utilizza la classe Java `KeyGenerator`, che fornisce la funzionalità di generatore di chiavi segrete. In particolare, in questo caso, per consentire il calcolo delle funzioni HMAC utilizza l'algoritmo `HmacSHA256`.

In sostanza, il puzzle contiene l'output dell'hash applicato ad h e i $m - k$ bit di h e il client deve trovare i k bit rimanenti di h con una complessità computazionale pari a $O(2^{k-1})$. Naturalmente, la premessa qui è che il client non abbia un modo più veloce per trovare i k bit mancanti, tranne che effettuando una ricerca per brute-force. La lunghezza k determina quante operazioni di hash debbano essere eseguite dal client, quindi determina la difficoltà del puzzle.

Una volta che il server ha creato il puzzle puz, risponde con il messaggio R_1, che contiene le seguenti informazioni:

$$R_1 = \{I_1, puz, t_{now}\}$$

dove puz è il puzzle generato in precedenza.

Dopo aver ricevuto il puzzle, se il client deve risolvere un puzzle, calcola la soluzione del puzzle *sol* effettuando un attacco brute-force alla ricerca di un valore che soddisfi

$$hash(sol \| h2) = h',$$

altrimenti salta alla fase del messaggio I_3. Il metodo che offre la risoluzione del puzzle è il metodo `resolvePuzzle()` mostrato di seguito.

```
public byte[] resolvePuzzle() {
        try {
                BigInteger h2Big = new BigInteger(h2);
                BigInteger bigI, hBig;
                MessageDigest messageDigest =
                   MessageDigest.getInstance("SHA-256");
                byte[] hpTry;

                for(int i = 0; i < Math.pow(2,nBitPuzzle); i++) {
                        bigI = BigInteger.valueOf((long)i);
                        hBig = h2Big.add(bigI);
                        hpTry =
                           messageDigest.digest(hBig.toByteArray());
                        if(Arrays.equals(hpTry, hp)) {
                                h = hBig.toByteArray();
                                return h;
                        }
                }
        }catch(Exception e) {
                e.printStackTrace();
        }
}
```

Una volta che il client ha trovato la soluzione *sol* che soddisfa l'uguaglianza, invia tale soluzione insieme alla sua richiesta iniziale al server con il messaggio I_2, vale a dire:

$$I_2 = \{R_1, sol\}$$

Quando il server riceve la soluzione del puzzle dal client, controlla prima il tempo di scadenza del puzzle t_{exp} e ignora la richiesta del client se $t_{exp} < now$. Quindi, verifica che il codice di integrità del messaggio h_s sia valido e che $hash(sol \| h_2) = h'$; il server risponde con un messaggio che asserisce se la connessione è stata accettata o meno nel messaggio R_2.

Se la connessione viene accettata dal server o il client non è tenuto a risolvere un puzzle, il client invia un terzo messaggio I_3 in cui invia il nome della risorsa che sta cercando. Quindi, il server prende tali informazioni da un database e risponde con la risposta nel messaggio R_3.

4.4.1 Client

Per testare le funzionalità del server, è stata anche progettata una classe di client che potesse scambiare messaggi con il server. Ogni client implementa il protocollo

di comunicazione con il server e riporta il tempo necessario per terminare la comunicazione. Ogni client eredita dalla classe `Thread`, in modo che uno qualsiasi di essi possa essere eseguito come thread indipendente; inoltre ciascuno di essi comunica con il server con un proprio socket inizializzato al momento della loro creazione. Di seguito è riportato il codice del metodo `run()` di un singolo client.

```
public void run() {
       long t1 = System.currentTimeMillis();
       while(finish == false && nTry > 0) { //retry if timeout
          expires
              try {
                     byte[] puzzleSolution;
                     String confirm = "OK";

                     sendRequest(); //send access request
                     p.receivePuzzle(ds); //receive the puzzle
                        and the port of the server connected to
                     if(p.getNBitPuzzle() != -1) {
                            System.out.println("Trying to acced
                               with puzzle difficulty =
                               "+p.getNBitPuzzle());
                            puzzleSolution = p.resolvePuzzle();
                            //System.out.println("Resolved
                               puzzle of "+p.getNBitPuzzle()+"
                               bits difficulty");
                            sendBytes(puzzleSolution); //send
                               the found resolution
                            confirm = receiveString(); //receive
                               the acceptance from the server
                     }else //no puzzle resolution required
                            System.out.println("Acceding without
                               puzzle");

                     if(confirm.equals("OK")) {
                            //System.out.println("Connection
                               accepted");
                            sendBytes("Item1".getBytes("UTF-8"));
                               //send the resource request
                            System.out.println("Resource
                               received: "+receiveString());
                            long t2 = System.currentTimeMillis();
                            System.out.println("Time legitimate
                               client:
                               "+((double)(t2-t1))/1000);
                     }else
                            System.out.println("Connection
                               refused");
                     finish = true;
```

```
                }catch(SocketTimeoutException e) {
                        nTry--;
                        e.printStackTrace();
                }
        }
        ds.close();
}
```

Come si può notare dal codice, il client inizia il protocollo inviando la prima richiesta con il metodo `sendRequest()`; successivamente riceve la risposta R_2 del server e risolve un puzzle con il metodo `resolvePuzzle()` se necessario. Successivamente invia l'identificativo della risorsa richiesta e la stampa a video con il tempo totale trascorso dall'inizio della comunicazione.

Un punto da notare è che ogni client tenta di eseguire il protocollo per tre volte, poiché la comunicazione client-server utilizza il protocollo di trasporto UDP, quindi è necessario prendere in considerazione le possibilità di perdite di pacchetti o congestioni di rete. Per questo motivo, per ogni socket è impostato un timer che indica il tempo massimo di attesa chiamando il metodo `receive()`.

4.4.2 Database

Il database viene utilizzato per archiviare le informazioni a cui i client desiderano accedere. La sua presenza permette di valutare i diversi tempi necessari per accedere ad una risorsa se ad un cliente è richiesto o meno per risolvere un puzzle. Per questo motivo, non è un requisito fondamentale progettare uno schema complesso del database, poiché l'unico fattore su cui si concentra la soluzione è il tempo necessario per accedere ad una risorsa. Esso consiste in una sola tabella contenente informazioni salvate come valori alfanumerici.

Il database è gestito tramite una connessione MySQL, la quale adotta il protocollo TLS per lo scambio di messaggi tra server e database, offrendo quindi autenticazione, integrità dei dati e confidenzialità. Le operazioni necessarie per comunicare con il database sono fornite dalla classe `Database`, che crea la connessione con il server utilizzando i driver JDBC e propone il metodo `getItem()`, che riceve il nome della risorsa richiesta dal client e restituisce una stringa contenente tutte le informazioni su quella risorsa. Il codice è mostrato di seguito.

```
public String getItem(String name) {
        try {
                //Statements allow to issue SQL queries to the
                   database
                statement = connection.createStatement();
                // Result set get the result of the SQL query
                resultSet = statement
                    .executeQuery("SELECT * FROM items WHERE
                       Name='"+name+"';");
                resultSet.beforeFirst();
                if(resultSet.next()) {
```

```
                    String toSend = resultSet.getInt(1)+"
                       "+resultSet.getString(2)+"
                       "+resultSet.getString(3)+" "
                           +resultSet.getInt(4)+"
                              "+resultSet.getDouble(5);
                    return toSend;
            }
        }catch (Exception e) {
                e.printStackTrace();
        }
}
```

Capitolo 5

Risultati

Questo capitolo presenta i metodi utilizzato per valutare la soluzione di difesa proposta e i risultati ottenuti dopo gli esperimenti. Il threat model da cui il server deve essere difeso è quello introdotto nella Sezione 3.2.

5.1 Sistema di analisi e metriche di valutazione

5.1.1 Sistema di analisi

Per garantire che la soluzione sviluppata funzioni correttamente durante un attacco, è stato creato un ambiente costituito da un server, numerosi client legittimi e un attaccante. Tutti i client sono in esecuzione sulla stessa macchina e si differenziano controllando il numero della porta remota da cui essi effettuano l'accesso al server.

Per valutare le prestazioni del server, sono state utilizzate due differenti classi di client: i client legittimi e quelli malevoli, i quali tentano di ottenere l'accesso al server in modo persistente. Ogni client legittimo accede al server da una porta diversa, mentre i client illegittimi accedono utilizzando sempre la stessa porta remota. Il server tiene traccia delle richieste inviate da tutti i client, in particolare anche la porta da cui esse sono inviate, quindi può facilmente verificare se un client è legittimo confrontando il numero di accessi effettuati dalla porta da cui si sta ricevendo la richiesta e la soglia del tasso d'arrivo di richieste desiderata per evitare la congestione del server, impostata al suo avvio.

Il test di simulazione crea un numero elevato di client legittimi, ognuno dei quali effettua una connessione al server e viene eseguito in un thread dedicato. Dopo aver creato tutti i client, il test attende la fine di tutti i thread precedentemente creati chiamando il metodo `join()` della classe `Thread`, che consente di interrompere l'esecuzione del thread principale fino a quando tutti i thread secondari attualmente in esecuzione non hanno terminato il loro lavoro. Il codice del test è mostrato di seguito.

```
public static void main(String[] args) {
        try {
                int minT = 0, maxT = 1000;
```

```
                long t1, t2;
                List<Client> clients = new ArrayList<>();

                t1 = System.currentTimeMillis();
                for(int i = 0; i < 1000; i++) {
                        Client c = new
                           Client(InetAddress.getByName(serverName),
                           serverPort);
                        clients.add(c);
                        c.start();
                        Thread.sleep((int)(Math.random()*(maxT-minT+1)+minT));
                }
                for(Client c : clients)
                        c.join();
                t2 = System.currentTimeMillis();
                System.out.println("Average execution time:
                   "+((double)(t2-t1))/1000);
        }catch(Exception e) {
                e.printStackTrace();
        }
}
```

Come si può vedere dal codice, la simulazione adottata in questa trattazione utilizza un numero di 1000 client. Questo numero deve essere deciso in base alla macchina in cui è in esecuzione il server e alla potenza della sua CPU. Se la simulazione venisse eseguita in una macchina diversa, il numero di client potrebbe cambiare e i risultati numerici potrebbero subire alcune modifiche, ma le considerazioni finali ottenute dalla simulazione sarebbero comunque le stesse in tutti i casi.

I client illegittimi vengono creati utilizzando la stessa classe `Client` introdotta nella Sezione 4.4.1, ma in questo caso l'esecuzione non si interrompe mai, rendendo persistente l'accesso al server utilizzando un ciclo infinito.

Vengono inoltre presi in considerazione questi dettagli aggiuntivi:

- Sono considerati due tipi di attacco: attacchi flooding e attacchi replay.
- L'attaccante nell'attacco replay puzzle tenta di riutilizzare una soluzione valida trovata precedentemente a una velocità preconfigurata fino alla scadenza del puzzle.
- La soglia per cui un client è considerato illegittimo è impostata su 100 accessi/minuto.

Tutte le valutazioni delle prestazioni vengono eseguite su una macchina fisica con seguente software e hardware:

- **Tipo**: architettura x64;

- **CPU**: Intel(R) Core(TM) i7-5500U, Dual-core @ 2.40GHz;
- **RAM**: 8 GB DDR3 @ 1600 Hz;
- **SO**: Windows 10.

5.1.2 Metriche di valutazione

Sono state adottate tre metriche di valutazione: tempo medio di completamento di una singola richiesta, numero di pacchetti inviati dai client malevoli e percentuale di richieste legittime negate. Il tempo medio di completamento viene calcolato registrando il tempo trascorso tra l'invio di una richiesta e la ricezione della risposta contenente la risorsa ricercata, il quale comprende anche il tempo speso per risolvere il puzzle, per tutte le richieste completate di tutti i client legittimi e prendendone infine la media aritmetica. Il numero di pacchetti inviati dai client malevoli è il numero di pacchetti inviato da una stessa porta e deve diminuire quando il server percepisce che è sotto attacco, ovvero quando viene superata la soglia del tasso d'arrivo delle richieste. La percentuale di richieste legittime negate viene calcolata dividendo il numero totale di richieste legittime negate dal servizio per il numero totale di richieste legittime inviate.

5.2 Risultati

Questa sezione presenta i risultati ottenuti attraverso la simulazione proposta antecedentemente e sulla base dei parametri di valutazione descritti nel precedente paragrafo.

Figura 5.1. Tempo di accesso

La Figura 5.1 mostra il tempo necessario per accedere a una singola risorsa in base al numero di accessi di un singolo client (quelli legittimi vengono eseguiti

nella stessa applicazione in modo che il loro numero di accessi sia cumulativo, ma ognuno di essi viene contato come se ad accedesse una singola volta). Per i clienti legittimi il tempo richiesto rimane costante, nell'intorno di 4/5 millisecondi, perché non sono obbligati a risolvere un puzzle, essendo il loro numero di accesso al di sotto della soglia del tasso d'arrivo impostato sul server. Al contrario, i client illegittimi, dopo un transitorio in cui non sono ancora riconosciuti come client malevoli e non devono risolvere un puzzle, sono obbligati a trovare una soluzione di un puzzle ogni volta che inviano una nuova richiesta al server e la difficoltà da risolvere continua ad aumentare fino a quando l'attaccante non riesce a trovare una soluzione in un tempo ragionevole. Ad esempio nel caso mostrato nella figura 5.1 l'attaccante impiega più di 2 minuti per trovare una soluzione puzzle dopo aver inviato più di 100 richieste di accesso.

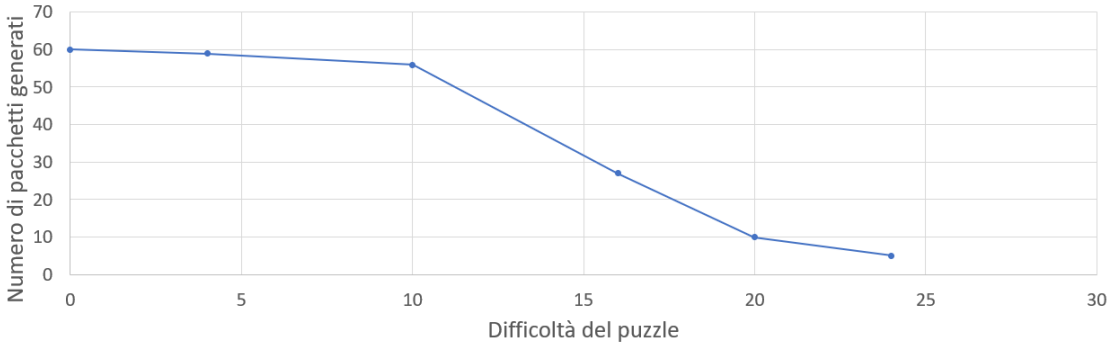

Figura 5.2. Rate di generazione dei pacchetti

La Figura 5.2 mostra il numero di pacchetti generati da un client malevolo che intende accedere in modo persistente al server. Questa immagine è stata catturata con un singolo client che tenta di accedere al server con un loop infinito. All'inizio, quando la difficoltà del puzzle è troppo bassa per impedire l'accesso degli attaccanti, il rate rimane piuttosto alto fino a quando la difficoltà raggiunge i 10 bit. Da questo momento, il numero di pacchetti inviati dallo stesso client diminuisce drasticamente, raggiungendo meno di 10 pacchetti quando la difficoltà del puzzle è impostato su 24, il che significa che il tasso di pacchetti dell'attaccante è stato ridotto di oltre il 91%.

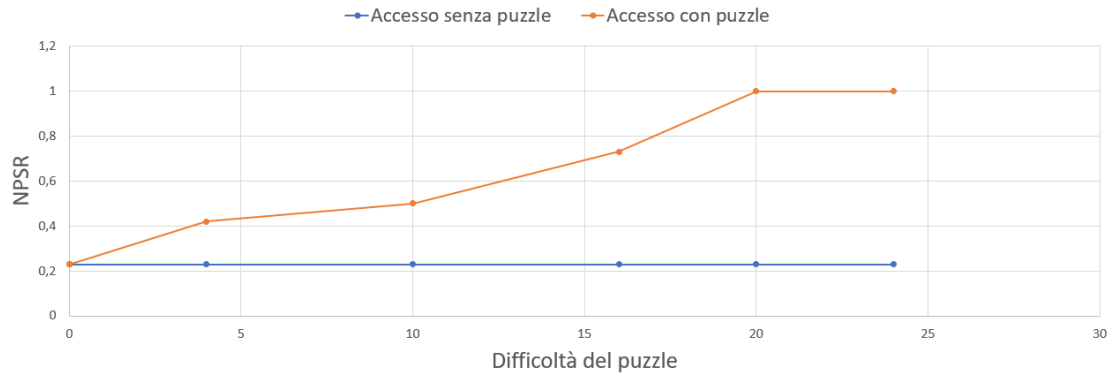

Figura 5.3. NPSR per client legittimi

La figura 5.3 mostra il Normal Packet Survival Ratio (NPSR) per i clienti legittimi di questo esperimento. L'NPSR indica il numero di pacchetti legittimi che effettivamente raggiungono il server diviso il numero totale di pacchetti generati dai client legittimi. Fondamentalmente, rappresenta la percentuale di richieste legittime accettate. Man mano che il livello di difficoltà aumenta per i client illegittimi, il carico computazionale sul server viene alleviato perché i pacchetti appartenenti a quei client vengono eliminati, invece i pacchetti appartenenti a quelli legittimi non vengono eliminati e le loro richieste sono tutte soddisfatte dal server, in modo che l'NPSR raggiunga 1; ciò sta a significare che vengono elaborate tutte le richieste legittime. In questo esperimento, dalla figura 5.3 è emerso che la difficoltà di puzzle ottimale per difendersi dagli attacchi DDoS è di 20 bit. Con questa difficoltà, ci sono pochissimi pacchetti persi e il throughput dei client rimane piuttosto elevato. Senza puzzle, solo il 23% delle richieste dei client sono state ricevute e accettate dal server. Con la soluzione proposta al livello di difficoltà 20, vi è meno dell'1% di perdita di pacchetti e oltre il 99% delle richieste legittime è soddisfatto. Inoltre, grazie al fatto che ai client legittimi non è richiesto di risolvere puzzle, il tasso di generazione pacchetti di ciascuno di essi è costante, al contrario il tasso dei client illegittimi si riduce significativamente, come mostra la Figura 5.2.

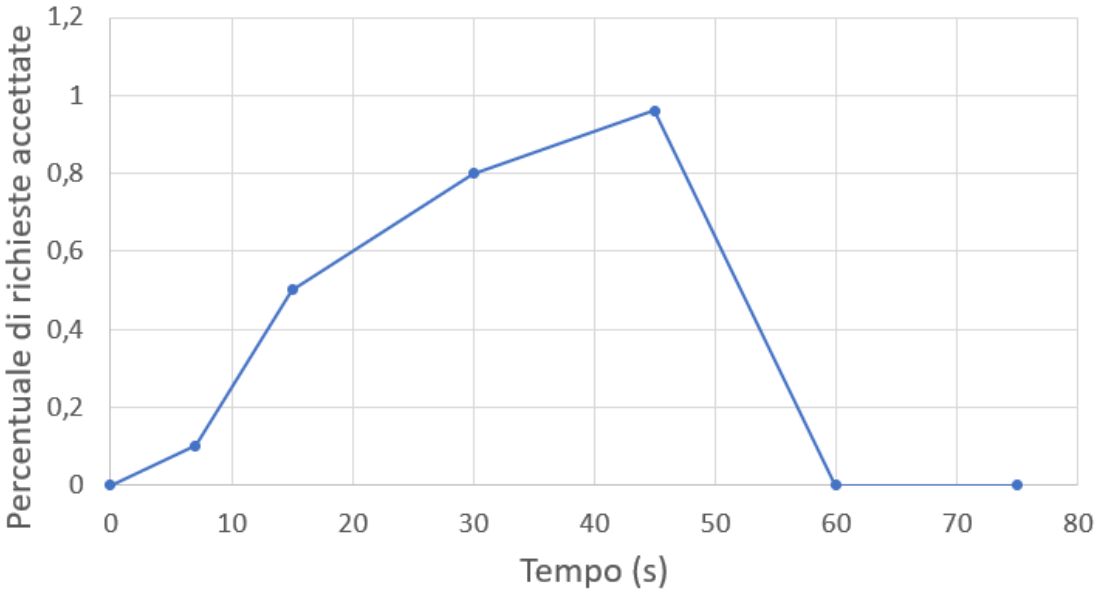

Figura 5.4. Reazione agli attacchi replay

Gli effetti durante un attacco replay sono mostrati in figura 5.4. Questo test illustra l'utilizzo della CPU del server quando un client utilizza la stessa soluzione di puzzle valida per poter ottenere l'accesso. Tale valore è ottenuto chiamando il metodo `getSystemCpuLoad()` della libreria Java. L'istante iniziale corrisponde al momento in cui un utente accede con una soluzione di puzzle valida. Da quel momento, esso continua ad accedere al server utilizzando sempre la stessa soluzione, essendo corretta. L'utilizzo del server continua ad aumentare fino a raggiungere il valore di picco 1, che indica l'utilizzo totale della CPU. Quando però il tempo trascorso dalla risoluzione del puzzle supera quello della sua validità (in questo caso di 60 secondi), il client deve di nuovo risolvere un altro puzzle per poter vedere

soddisfatta una sua richiesta, richiedendo un quindi ulteriore tempo di calcolo. Grazie a questo, dopo il tempo di validità del puzzle, il valore dell'utilizzo della CPU torna a valori di bassa attività.

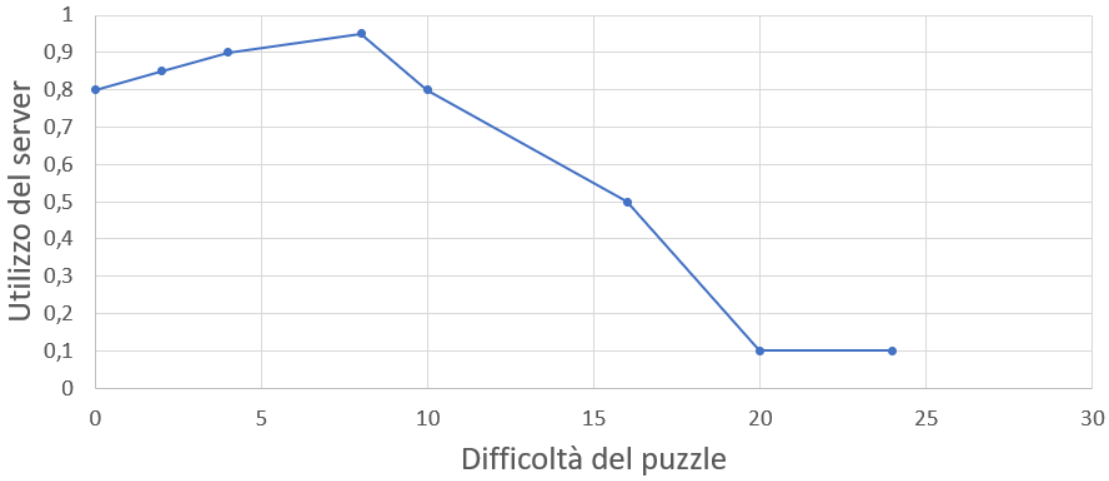

Figura 5.5. Utilizzo del server

Per concludere, la figura 5.5 mostra l'utilizzo del server durante un attacco. Come si evince dal grafico, l'utilizzo del server assume valori prossimi all'80% quando i puzzle non sono ancora attivati o quando essi hanno ancora un valore basso. Con l'aumentare della difficoltà, il server impone ai client malintenzionati di risolvere un numero sempre più elevato di funzioni hash, per cui non deve più servire le numerose richieste in arrivo, permettendo di avere quindi un utilizzo della CPU relativamente basso.

Capitolo 6

Lavoro futuro e conclusioni

In questa tesi, sono state illustrate e studiate varie tecniche di mitigazione per attacchi DoS e DDoS a livello applicazione. In particolare, è stato studiato più approfonditamente l'argomento dei client puzzle il quale è poi stato applicato al livello applicazione dello stack OSI. Nella letteratura recente, il concetto di client puzzle è stato analizzato nel dettaglio e ha mostrato risultati soddisfacenti nel mitigare gli effetti di un attacco DoS. Questa tesi ha promosso ulteriormente la ricerca svolta in quest'area progettando, implementando e sperimentando un nuovo protocollo di client puzzle.

L'implementazione ha consentito esperimenti su sistemi reali e in scenari di attacco tipici. Gli esperimenti hanno avuto successo in quanto hanno dimostrato che la soluzione trovata è in grado di preservare la disponibilità del server e gli effetti degli attacchi DoS sono stati attenuati in modo che il server rimanesse disponibile ai client legittimi quando un attacco era in corso.

Mitigare gli attacchi DoS è sempre stato un argomento di ricerca impegnativo e continuo nella sicurezza delle reti. La ricerca che è stata presentata in questa tesi potrebbe sempre essere approfondita per apportare miglioramenti o nuove scoperte. La sezione seguente presenta un riepilogo sul lavoro futuro in questo ambito e sulla progettazione in generale dei protocolli di client puzzle.

6.1 Lavoro futuro

Un punto importante da tenere a mente nella progettazione di un protocollo di puzzle è che il tempo di risoluzione di un puzzle dipenderà sempre dalla potenza computazionale della macchina su cui esso si sta risolvendo. In un tale schema, un client più potente avrà sempre più probabilità di trovare una soluzione rispetto a client meno potenti. Questo inconveniente è comune alla maggior parte dei protocolli di puzzle. Come parte del lavoro futuro, si può tentare di esaminare più approfonditamente metodi in grado di regolare il livello di difficoltà del puzzle per ogni potenziale cliente. Adattando il livello di difficoltà per ogni client, si può impedire ai client più potenti (possibilmente dannosi) di avere un vantaggio sugli altri client. Questo concetto è noto come *equità*. La nozione di equità è un importante problema di ricerca nei client puzzle, perché una protezione più efficace contro gli

attacchi DoS può essere fornita distribuendo puzzle più difficili ai client che possiedono maggiori capacità computazionali. In tutti i protocolli di client puzzle, i puzzle possono diventare troppo difficili da risolvere per i client legittimi più deboli, facendo sì che i clienti più deboli impieghino più tempo a risolvere un puzzle e possano avere difficoltà nell'accedere ai servizi richiesti. Non essere in grado di raggiungere l'equità porta a un problema di disparità di risorse.

Un altro problema è che, sebbene i metodi di mitigazione DoS basati su puzzle siano promettenti, richiedono calcoli costosi da eseguire sulle macchine dei client e le risorse per la risoluzione dei puzzle non servano a niente altro che garantire l'accesso al server. Più specificamente, i calcoli del puzzle non danno luogo a soluzioni a problemi significativi, né svolgono un'attività utile che può in qualche modo dare dei vantaggi ai client. Il compito più comunemente assegnato ai client nei protocolli di puzzle è il calcolo di milioni di hash crittografici e tali calcoli non costituiscono una funzionalità che fornisce utilità a qualche processo. Un miglioramento per evitare questo dispendioso problema di calcolo sarebbe renderli attività utili a applicazioni o servizi che forniscono funzionalità significative agli utenti.

In tutti i protocolli di puzzle sviluppati fino ad oggi, il tempo di risoluzione di un puzzle è sempre relativo alla potenza di elaborazione della CPU del client. Il lavoro futuro con i client puzzle prevede la progettazione di nuovi e diversi tipi di puzzle. Come accennato in precedenza nel Capitolo 2, i puzzle legati alla memoria sono stati proposti come un approccio migliore rispetto ai tradizionali puzzle basati sul calcolo. Un vantaggio dei puzzle legati alla memoria è che il tempo di risoluzione per un puzzle non ha una distribuzione così ampia come i puzzle tradizionali tra macchine con diversa capacità computazionale. In altre parole, avere più memoria potrebbe non aiutare a migliorare le possibilità di risolvere il puzzle più velocemente rispetto a un client più debole con meno memoria. Un interessante argomento di ricerca sarebbe quello di modificare lo schema presentato in questa tesi con puzzle legati alla memoria e quindi simulare attacchi con attaccanti e con client di diversa capacità per determinare se i puzzle legati alla memoria apporterebbero un miglioramento significativo.

Un'ultima considerazione è che nel capitolo 5 sono stati mostrati i risultati ottenuti durante un attacco flooding. Tuttavia, si potrebbero effettuare prove in altri ambienti con questo protocollo per testarne la scalabilità. Le simulazioni potrebbero essere fatte simulando questo protocollo in una rete realistica con molti più aggressori e client legittimi. Simulazioni come questa migliorerebbero notevolmente i risultati raccolti.

6.2 Conclusioni

I client puzzle sono un nuovo metodo per difendersi dagli attacchi DoS, ma la loro scalabilità è un problema che trarrebbe grandi vantaggi da ulteriori ricerche. Se un utente malintenzionato ha una grande quantità di risorse e potenza di calcolo possedendo un gran numero di zombie, l'efficacia dei client puzzle potrebbe non essere così grande come inizialmente previsto. Tuttavia, come ha dimostrato questa tesi, questi protocolli hanno migliorato la resilienza di un server durante alcuni tipi

di attacchi. L'idea alla base dei protocolli di puzzle, o un protocollo proof-of-work, è un argomento promettente che potrebbe portare a miglioramenti e progressi nel networking e nella sicurezza delle reti nel futuro.

Come accennato nel primo capitolo, la mitigazione è solo un aspetto di una contromisura DoS. Per sconfiggere gli attacchi DoS è necessario studiare varie vulnerabilità nelle reti di calcolatori e progettare una serie di meccanismi che possano impedire la comparsa di questi attacchi. Tuttavia, la vera chiave per difendere i sistemi informatici da attacchi attraverso la rete sono la vigilanza e la consapevolezza costanti. L'elaborazione di contromisure aiuterà sempre a contrastare gli attacchi noti, ma poiché gli attacchi DoS differiscono sempre l'uno dall'altro con delle proprie peculiarità, combattere gli attacchi DoS richiede una conoscenza completa e una comprensione tecnica approfondita dei problemi fondamentali della sicurezza dei sistemi informatici e delle reti. Oltre alle tecniche di mitigazione, sconfiggere gli attacchi DoS implica anche impedire che le macchine vengano convertite in gestori o zombie. Ciò richiede un rilevamento accurato degli attacchi e quindi una metodologia efficiente per eseguire analisi a seguito di un attacco. La combinazione di questi metodi si rivelerà il mezzo più efficace per combattere la minaccia degli attacchi DoS e per consentire a Internet di diventare un mezzo più affidabile, utile e sicuro.

Appendice A

Manuale utente

In questa sezione sono spiegate le procedure necessarie all'avvio del programma e all'ottenimento dei dati delle simulazioni.

A.1 Classi di esecuzione

Il progetto è scritto in linguaggio Java, per cui è richiesta l'installazione di una Java Virtual Machine sulla macchina di esecuzione. Le versioni richieste per poter eseguire adeguatamente il progetto sono le versioni Java 1.8 o superiori. Per poter effettuare correttamente una simulazione è necessario dover eseguire il server, i vari client legittimi e i client illegittimi. Per poter eseguire ciascuna di questa entità sono state utilizzate tre classi Java che permettono di iniziare i processi a loro associati. Le tre classi eseguibili sono le seguenti:

- `ServerMain`: contiene solamente un metodo `main` in cui si crea un oggetto `Server` (si veda la Sezione B) e si invoca il suo metodo `StartServer` che avvia il processo legato al server;
- `ClientMain`: nel metodo `main` viene creato un numero elevato di utenti considerati legittimi per le prove e vengono salvati in un `List<Client>` (le caratteristiche della classe `Client` si trovano nella Sezione B). Ciascuno di essi eredita dalla classe `Thread` e viene eseguito in un thread dedicato;
- `MaliciousClient`: crea un client illegittimo che invia un numero di richieste infinite al server. Esso ha le stesse caratteristiche della classe `Client` (si vede la Sezione B) con l'unica differenza della durata del ciclo, in questo caso infinito.

A.2 Database

Il server effettua delle letture da un database per poter le informazioni inviate ai suoi client. Il database utilizzato è un database MySQL e per poter interagire con esso è necessario aver installato una versione di MySQL 5.0 o superiore.

Per poter creare il database e gestire le tabelle è necessario disporre di uno strumento in grado di provvedere gestione, modellazione dati, creazione e manutenzione di database MySQL. Per questo scopo si consiglia di utilizzare una versione di MySQL Workbench 8.0 o superiore, tenendo in conto che possono essere utilizzati anche altri strumenti (ad esempio XAMPP).

Una volta installato ed avviato MySQL Workbench, le operazioni da eseguire per poter creare correttamente il database utilizzato dalle simulazioni sono le seguenti:

- Nella pagina iniziale d'avvio cliccare sul pulsante *New connection* da cui si aprirà una finestra per generare una nuova connessione;
- Assegnare un nome alla connessione e completare i campi *username* e *password* con i dati utilizzati in fase di installazione;
- Premere il pulsante *OK* per portare a termine la creazione della connessione; essa apparirà nella schermata principale da cui è possibile accedervi;
- Aprire la connessione; dalla barra degli strumenti estendere il menu a tendina sulla voce *Server* e poi selezionare *Data Import*;
- Nella nuova finestra selezionare la voce *Import from Self-Contained File*, quindi aggiungere il file `database.sql` presente nella cartella compressa;
- Scegliere il default target schema per decidere in quale database salvare le nuove tabelle;
- Andare nella scheda *Import Progress* e premere quindi sul pulsante *Import*;
- A questo punto il database è stato creato; per poter vedere comparire le tabelle nel menu di sinistra selezionare la connessione e premere sul tasto *Refresh*.

Una volta creato il database, è necessario disporre di un connettore in modo che il server possa interagire con esso. Il connettore da utilizzare è il *mysql connector* versione *8.0.19* (disponibile al link `https://dev.mysql.com/downloads/connector/j/`). Per poter essere reso visibile esso deve essere aggiunto al build path del progetto. In caso si stia utilizzando Eclipse come ambiente di programmazione, i passi sono i seguenti:

- Dalla barra degli strumenti premere su *Project* e selezionare *Properties*, quindi aprire la sezione *Java Build Path*;
- Aprire la scheda *Libraries*;
- Premere sul pulsante *Add External JARs* e quindi selezionare il connettore MySQL;
- Terminare la procedura con il pulsante *Apply and Close*.

A questo punto anche il connettore è stato aggiunto al progetto ed il server è pronto per essere avviato.

Appendice B

Manuale del programmatore

In questa sezione viene introdotta l'analisi del codice utilizzato per sviluppare il progetto, spiegando nel dettaglio le strutture dati utilizzate e le funzionalità di ciascun metodo.

Il progetto è scritto usando il linguaggio Java e comprende lo sviluppo di sette classi principali:

- `Server`: rappresenta l'entità server e racchiude tutte le sue funzionalità;
- `Client`: rappresenta un client ed include i metodi che servono a richiedere un servizio al server;
- `ClientStructure`: rappresenta l'entità di un client salvata lato server;
- `MaliciousClient`: rappresenta un client malevolo;
- `Puzzle`: include tutte le funzionalità relativa alla costruzione, invio e verifica di un puzzle. Viene usata per creare gli oggeti puzzle che vengono utilizzati sia dalla classe `Server` sia dalla classe `Client`;
- `Database`: offre le funzioni di creazione di una connessione ad un database e di lettura dei dati da parte del server;
- `Functions`: contiene alcuni metodi statici utili all'esecuzione di routine comuni sia del server sia dei client.

Segue ora la presentazione delle caratteristiche di ciascuna delle classi precedenti.

B.1 Server

La classe `Server` rappresenta l'oggetto server e ha i seguenti attributi:

- `int serverPort`: è il numero di porta su cui il server è in ascolto;

- `Database database`: rappresenta l'oggetto database con cui il server ha una connessione aperta;
- `DatagramSocket socket`: è il socket su cui arrivano le richieste dei client;
- `Map<String,ClientStructure> clients`: è una mappa che serve a mantenere le informazioni riguardanti lo stato di ciascun client che tenta di accedere al server (per maggiori informazioni sulla classe `ClientStructure` si rimanda alla sezione B.4);
- `Map<String,Integer> access`: è una mappa che per ciascun client (rappresentato come `String` ottenuta concatenando l'indirizzo con la porta remota di ciascun client) mantiene il suo numero di accessi;
- `int limitRequest`: rappresenta la soglia di accessi massimi consentiti.

I metodi offerti sono i seguenti:

- `void startServer()`: è il metodo principale della classe, che viene chiamato all'avvio del server e che svolge tutte le attività principali di ricezione di datagram, salvataggio delle informazioni e invio dei messaggi ai client, all'interno di un ciclo `while` infinito. Ad ogni passo del ciclo riceve un datagramma e utilizza l'indirizzo di origine e il numero di porta di origine per controllare lo stato di un client nella mappa `clients` prima introdotta e quindi decidere quale fase del protocollo dover applicare;
- `int generateNBitPuzzle(int nAccess)`: calcola la difficoltà del puzzle per un client in base al numero dei suoi accessi `nAccess` e lo ritorna come intero positivo. La difficoltà è calcolata come rapporto tra il numero di accessi `nAccess` e la soglia preimpostata `limitRequest`. Ritorna -1 nel caso non sia richiesto risolvere un puzzle;
- `void handleResourceRequest(String request, InetAddress clientAddress, int remotePort)`: accede al database chiedendo una riga identificata dall'ID `request` e lo invia al client identificato dal suo indirizzo e dalla sua porta remota.

B.2 Client

La classe `Client` rappresenta un oggetto client ed eredita dalla classe `Thread`.

Contiene i seguenti attributi:

- `InetAddress serverAddress`: indirizzo del server a cui ci si vuole connettere;
- `int serverPort`: porta del server a cui inviare le richieste;
- `DatagramSocket ds`: socket utilizzato per inviare i datagrammi;

- `Puzzle p`: puzzle ricevuto dal server (per i dettagli della classe puzzle vedere la Sezione B.5);
- `int nTry`: numero massimo di tentativi di esecuzione del protocollo in caso di insuccesso;
- `boolean finish`: indicatore di terminazione del protocollo (`true` se il protocollo è terminato correttamente).

I suoi metodi sono:

- `void run()`: è il metodo ereditato dalla classe `Thread` e viene eseguito ogni volta che si invoca il metodo `start` su un nuovo client. Utilizza un ciclo `while` in cui si invia una richiesta al server, si risolve un puzzle (se richiesto) e si riceve la risorsa richiesta precedentemente richiesta al server. Il ciclo termina quando il protocollo finisce oppure quando il numero di tentativi massimi di esecuzione `nTry` è stato superato;
- `void sendRequest()`: invia la richiesta iniziale al server. Il messaggio inviato è costituito dalla concatenazione tra stringhe dell'indirizzo del server, dell'indirizzo del client e della risorsa richiesta (rappresentata come `String`).

B.3 MaliciousClient

La classe `MaliciousClient`, che modella un client malevolo, ha le stesse funzionalità della classe `Client` (vedere la Sezione B.2 per maggiori dettagli); l'unica differenza è che il suo ciclo `while` è infinito e non termina mai.

B.4 ClientStructure

Rappresenta un'entità client, contenente tutte le sue informazioni, salvata nel server per mantenerne lo stato. I metodi che contiene sono solamente dei metodi `get` dei suoi seguenti attributi:

- `InetAddress clientAddress`: l'indirizzo di un client;
- `int remotePort`: la porta remota associata ad un client;
- `Puzzle p`: il puzzle generato per quel client;
- `boolean connected`: indica se il client in questione ha inviato una soluzione del puzzle corretta;
- `Timestamp timestamp`: rappresenta il tempo di arrivo di una richiesta di un client.

B.5 Puzzle

Questa classe fornisce tutte le funzionalità per la creazione, invio, ricezione e verifica di un puzzle. Essa viene utilizzata sia lato server sia lato client. I suoi attributi sono:

- `byte[] h`: rappresenta il risultato in bit del calcolo dell'HMAC della concatenazione tra indirizzo del client, porta del client e richiesta ricevuta dal server;
- `byte[] h2`: rappresenta i bit più significativi di `h`;
- `byte[] hp`: è ottenuto calcolando `hash(h)`;
- `Timestamp timestamp`: il tempo in cui è stato creato il puzzle;
- `int nBitPuzzle`: la difficoltà assegnata ad un puzzle;
- `SecretKeySpec puzzleKey`: la chiave utilizzata per cifrare un puzzle; utilizza come algoritmo di cifratura l'algoritmo AES.

I metodi della classe lato server sono:

- `void generatePuzzleKey()`: genera una chiave AES e la assegna all'attributo `puzzleKey`;
- `void generatePuzzle(byte[] data, int nBitPuzzle)`: genera un puzzle di difficoltà `nBitPuzzle` a partire dal vettore di byte iniziale `data` (ovvero la richiesta ricevuta dal client);
- `void sendPuzzle(DatagramSocket ds, InetAddress clientAddress, int remotePort)`: invia il puzzle utilizzando il socket `ds` all'indirizzo `clientAddress` e alla porta `remotePort`;
- `boolean checkPuzzle(byte[] v)`: controlla se la soluzione ricevuta di un puzzle è corretta. Ritorna `true` se la soluzione `v` ricevuta coincide con `h`;

I metodi della classe lato client sono:

- `void receivePuzzle(DatagramSocket ds) throws SocketTimeoutException`: riceve un puzzle dal socket `ds` e lo salva negli attributi `h2` e `hp`; lancia un'eccezione `SocketTimeoutException` se non si riceve un messaggio entro il tempo di scadenza impostato sul socket;
- `byte[] resolvePuzzle()`: ritorna la soluzione `h` di un puzzle ottenuta tramite brute force.

B.6 Database

La classe database fornisce le funzionalità per creare una connessione con un database MySQL e per effettuare letture da esso. Il suo unico attributo è il parametro `connection` della classe `Connection`, che rappresenta la connessione al database. Essa viene ottenuta chiamando il metodo statico `getConnection` della classe `DriverManager`.

Il suo unico metodo è il metodo `String getItem(String name)` che, attraverso una query che utilizza la classe `Statement`, ottiene una riga dal database identificata dall'ID `name` e la ritorna come una unica stringa al server.

B.7 Functions

È una classe che offre dei metodi per routine comuni a server e client e contiene la definizione di alcune costanti utili per la definizioni dei parametri di esecuzione. Le costanti salvate sono:

- `WAIT_TIME`: tempo di attesa massimo di ricezione di un messaggio da un `DatagramSocket`; oltre questo tempo viene lanciata una `SocketTimeoutException`;
- `N_TRY`: numero massimo di tentativi dell'esecuzione del protocollo in caso di insuccesso.

I metodi contenuti sono i seguenti:

- `byte[] calcHmacSha256(SecretKeySpec secretKey, byte[] message)`: calcola un HMAC del vettore `message` con l'algoritmo SHA-256 e la chiave `secretKey`;
- `String bytesToHex(byte[] hashInBytes)`: fornisce la rappresentazione in esadecimale come `String` del vettore `hashInBytes`.

Bibliografia

[1] Global DDoS threat landscape report Q1 2017. Technical report, Imperva, May 2017.

[2] 2019 global DDoS threat landscape report. Technical report, Imperva, December 2019.

[3] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Technol.*, 5(2):299–327, May 2005.

[4] D. P. Agrawal and B. Xie. Encyclopedia on ad hoc and ubiquitous computing: Theory and design of wireless ad hoc, sensor, and mesh networks. In *Security attacks and challenges in wireless sensor networks*, page 403. World Scientific, 2010.

[5] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. Dos-resistant authentication with client puzzles. In Bruce Christianson, James A. Malcolm, Bruno Crispo, and Michael Roe, editors, *Security Protocols*, pages 170–177, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[6] Adam Back. Hashcash - a denial of service counter-measure. September 2002.

[7] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. Policing congestion response in an internetwork using re-feedback. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 277–288, New York, NY, USA, 2005. Association for Computing Machinery.

[8] Drew Dean and Adam Stubblefield. Using client puzzles to protect tls. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, USA, 2001. USENIX Association.

[9] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 426–444, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[10] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[11] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[12] Bogdan Groza and Bogdan Warinschi. Cryptographic puzzles and dos resilience, revisited. *Des. Codes Cryptography*, 73(1):177–2072, October 2014.

[13] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. January 1999.

[14] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming ip packet flooding attacks. *SIGCOMM Comput. Commun. Rev.*, 34(1):45?50, January 2004.

[15] B. Laurie and Richard Clayton. Proof-of-work, proves not to work. 2004.

[16] X. Liu, X. Yang, and Y. Xia. Netfence: Preventing internet denial of service from inside out. *SIGCOMM'10 - Proceedings of the SIGCOMM 2010 Conference*, pages 255–266, August 2010.

[17] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, April 1978.

[18] J Mirkovic. and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. In *ACM SIGCOMM Computer Communications Review*, volume 34, page 48. April 2004.

[19] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. volume 1, pages 289–300, 10 2007.

[20] Wu-chang W. Feng, Edward Kaiser, and A. Luu. Design and implementation of network puzzles. In *Proceedings - IEEE INFOCOM*, volume 4, pages 2372 – 2382, April 2005.

[21] Xiaofeng Wang and Michael Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. pages 257–267, January 2004.

[22] XiaoFeng Wang and Michael K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, page 78, USA, 2003. IEEE Computer Society.

[23] Brent Waters, Ari Juels, J. Halderman, and Edward Felten. New client puzzle outsourcing techniques for dos resistance. pages 246–256, January 2004.