Sicurezza dei sistemi e delle reti[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2014/15

Lezione V: Scansioni

Attività di ricognizione

Per progettare difese o attacchi occorre partire da attività di **ricognizione** delle reti obiettivo.
Il **difensore** *dovrebbe* conoscere la "Cartografia di reti e servizi", ma non sempre è cosí...
L'**attaccante**:

- Social engineering, WHOIS, DNS, Google
- Scanning

Socket programming

Server

```
1  int sd, sd_current;
2          socklen_t size;
3  struct sockaddr_in sin, pin;
4
5  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) /* TCP */
6                  == -1) {perror("socket");exit(1);}
7
8  memset(&sin, 0, sizeof(sin));
9  sin.sin_family = AF_INET;
10 sin.sin_addr.s_addr = INADDR_ANY;
11 sin.sin_port = htons(PORT);
12
13 if (bind(sd, (struct sockaddr *) &sin, sizeof(sin))
14                 == -1){perror("bind");exit(1);}
15
16 if (listen(sd, 5)
17                 == -1) {perror("listen");exit(1);}
18
19 if (sd_current =
20                 accept(sd, (struct sockaddr *) &pin, &size)
21                 == -1) {perror("accept");exit(1);};
22
23 /* send/recv */
24
25 close(sd_current); close(sd);
```

Client

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
int sd;
struct sockaddr_in sin, pin;

memset(&pin, 0, sizeof(pin));
pin.sin_family = AF_INET;
if (inet_aton(argv[1], &pin.sin_addr)
    == 0) {perror("inet_aton");exit(1);};
pin.sin_port = htons(PORT);

if ((sd = socket(AF_INET, SOCK_STREAM, 0))
    == -1) {perror("socket");exit(1);}

if (connect(sd,(struct sockaddr *) &pin,
            sizeof(pin)) == -1) {perror("connect");exit(1);}

/* send/recv */

close(sd);
```

Ping

- ICMP: protocollo per scambiare messaggi di controllo e diagnostici. ping manda pacchetti ICMP che chiedono una risposta.
- Esistono programmi per ping (non solo ICMP) massivi (`hping`, `fping`, `nmap`).
- Spesso ICMP viene filtrato per evitare questo tipo di attività.

Traceroute

- traceroute usa i TTL per analizzare una rete.
- Inizia mandando un pacchetto (ICMP o UDP) con `TTL==1` e si aspetta una risposta ICMP `TTL exceeded`: il mittente sarà un router a distanza 1 hop.
- Si ripete con TTL crescenti finché non si riceve un reply dalla destinazione finale.

Port scanning

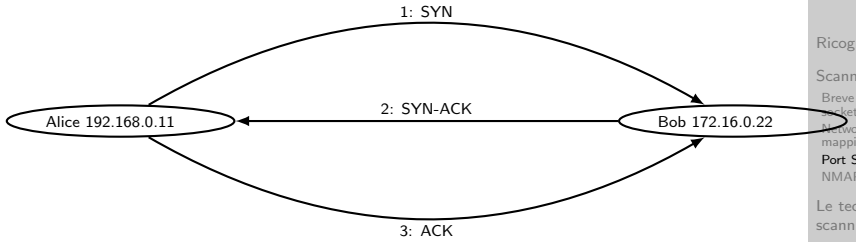
La conoscenza di quali *porte* sono accessibili (TCP o UDP) identifica i possibili canali di comunicazione:

- quali applicazioni monitorare
- quali canali sono utilizzabili in un attacco

Stato di una porta

open	Possibilità di connessione con un'applicazione (non necessariamente quella standard!)
closed	Accessibile, ma non c'è nessuna applicazione in ascolto
filtered	Appare *closed* ($\neg$ *open*) per **filtraggio** (del router, firewall, ecc.)

Nel caso di TCP



- Un SYN a porta chiusa → RST
- Un SYN-ACK → RST
- Un RST viene ignorato

UDP scan

UDP privo di *handshake*: un po' piú complicato

- lo stato è segnalato tramite ICMP
- lento, e sostanzialmente basato su timeout
- non molto affidabile, perché spesso ICMP è filtrato (p.es., solo x al minuto)

NMap

Nmap Scan	Command Syntax	Requires Privileged Access	Identifies TCP Ports	Identifies UDP Ports
TCP SYN Scan	`-sS`	YES	YES	NO
TCP connect() Scan	`-sT`	NO	YES	NO
FIN Scan	`-sF`	YES	YES	NO
Xmas Tree Scan	`-sX`	YES	YES	NO
Null Scan	`-sN`	YES	YES	NO
Ping Scan	`-sP`	NO	NO	NO
Version Detection	`-sV`	NO	NO	NO
UDP Scan	`-sU`	YES	NO	YES
IP Protocol Scan	`-sO`	YES	NO	NO
ACK Scan	`-sA`	YES	YES	NO
Window Scan	`-sW`	YES	YES	NO
RPC Scan	`-sR`	NO	NO	NO
List Scan	`-sL`	NO	NO	NO
Idlescan	`-sI`	YES	YES	NO
FTP Bounce Attack	`-b`	NO	YES	NO

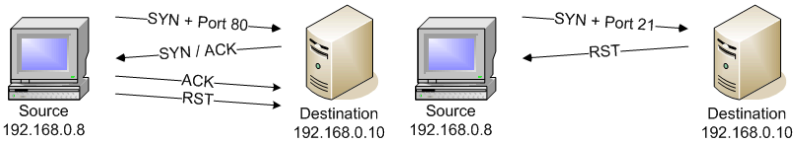
Riassumendo

- La conoscenza dei nodi e dei canali di comunicazione disponibili è fondamentale per attaccanti e difensori
- Documentazione, social engineering, WHOIS, DNS, Google...
- Scanning. Con Zmap (2013) è possibile esaminare l'intero spazio IPv4 in meno di un'ora (singola porta).

Connect

La modalità piú semplice è tentare una connessione (`connect()`)

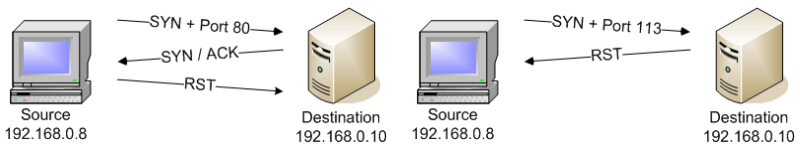
- non richiede privilegi particolari
- molto spesso l'evento viene registrato (e se la connessione avviene con lo stack standard il numero IP è quello reale)



SYN scan (half open)

Si risponde al SYN-ACK con un RST.

- È il metodo piú usato: veloce ed efficace
- Richiede i privilegi di root (non si può usare lo stack TCP standard)
- Piú difficile da "loggare"



TCP NULL, FIN, Xmas scan

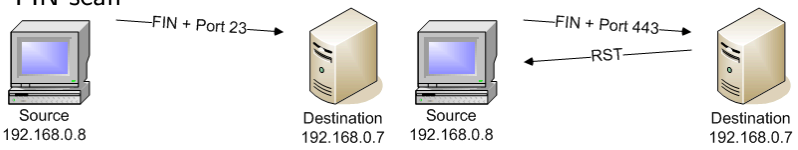
Si usano i flag in modo "creativo": invece di SYN, tutti gli altri in varie combinazioni; una porta chiusa risponde con un RST, una aperta invece li scarta (aspetta solo i SYN).

- Analoghi al SYN
- richiedono i privilegi di root
- ma ancora meno probabile una registrazione dell'evento

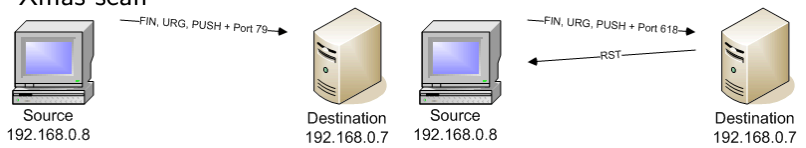
TCP NULL, FIN, Xmas scan

Attenzione però, se lo stack destinazione non è esattamente RFC 793 compliant, potrebbe agire in modo anomalo facendo apparire tutto chiuso

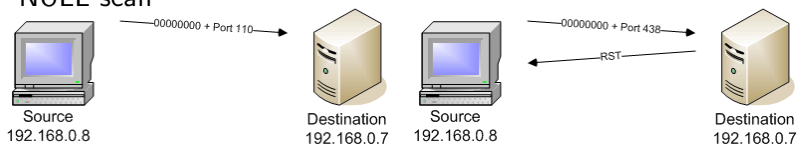
FIN scan



Xmas scan



NULL scan



Maimon scan: FIN-ACK; È possibile provare differenti combinazioni dei flag.

ACK scan, Window scan

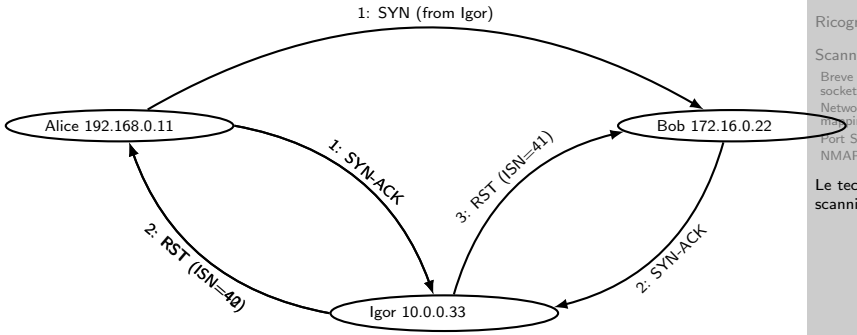
- Serve a determinare se c'è filtraggio.
- ACK: se non c'è filtraggio open e closed → RST
- se non c'è risposta o ICMP: filtered
- Window sfrutta la window size del RST ricevuto per distinguere fra open e closed (diversa in alcune implementazioni)

Idle scan

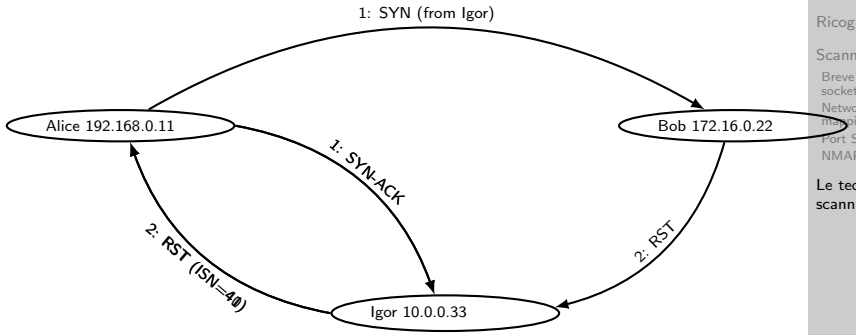
Lo scan viene compiuto da un nodo **inconsapevole** sfruttando il meccanismo di generazione degli ISN pacchetti , che talvolta è banalmente sequenziale.

- Un log conterrà l'IP della macchina "prestanome" (non è *spoofing* perché il nodo esiste e ha operato nel modo registrato)
- Il nodo deve essere idle, cioè non produrre traffico di rete **suo** durante lo scan
- Lo stack TCP deve incrementare banalmente gli ISN

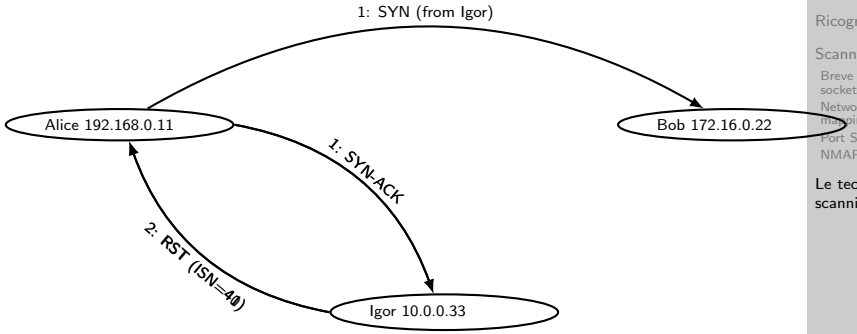
Idle scan con porta aperta



Idle scan con porta chiusa



Idle scan con porta filtrata



Riassumendo

- Sono note diverse tecniche per rilevare se una porta TCP è aperta
  - Semplice connessione
  - Pacchetti creati ad hoc
  - Idle scan