Report Tecnico: Sviluppo Tool Custom di Port Scanning per Infrastruttura Theta

1. Introduzione

Nell'ambito dell'incarico ricevuto dalla compagnia Theta per la messa in sicurezza e la progettazione della loro infrastruttura IT, è stata richiesta una fase approfondita di "Testing della Rete". L'obiettivo è garantire che l'infrastruttura sia ben progettata e sicura.

Una specifica fondamentale di questo audit riguarda la metodologia di scansione dei servizi: le linee guida del progetto vietano esplicitamente l'utilizzo di tool automatizzati preesistenti (come Nmap). È richiesto, invece, lo sviluppo di uno scanner in linguaggio Python proprietario che accetti in input un indirizzo IP e un range di porte da analizzare.

Questo report documenta lo sviluppo di due script Python, portscnRange.py e portscanALL.py, creati per mappare la superficie di attacco e identificare le porte aperte, chiuse o filtrate sui dispositivi di rete, come richiesto dai requisiti di progetto.

2. Analisi degli Script Sviluppati

Per soddisfare i requisiti e garantire flessibilità operativa, sono state sviluppate due varianti del tool di scansione.

A. portscnRange.py (Compliance ai Requisiti)

Questo script risponde direttamente alla richiesta formale dell'esercizio.

- **Funzionamento:** Il tool richiede all'utente l'IP target e un range di porte specifico (es. "20-100").
- **Logica:** Esegue il parsing della stringa del range, estraendo il porto minimo e massimo, e itera su questo intervallo.
- **Output:** Genera un elenco delle porte con il relativo stato (aperta/chiusa/filtrata), salvando i risultati in un file di log dedicato (scan_TARGET_MIN-MAX.log) per la reportistica finale.

B. portscanALL.py (Variazione Avanzata)

Questa è una variante potenziata progettata per un'enumerazione completa qualora non si conosca il range specifico dei servizi critici.

- **Funzionamento:** Tenta di scansionare l'intero spettro delle porte TCP (da 0 a 65535).
- **Meccanismo di Sicurezza:** Include una variabile MAX_SCAN_TIME (impostata a 60 secondi nel codice sorgente) che interrompe il ciclo se la scansione impiega troppo tempo. Questo previene che lo script rimanga "appeso" indefinitamente in caso di reti lente o dispositivi che droppano pacchetti.

3. Implementazione e Testing del Port Scanner Custom

In conformità con le specifiche di progetto, che vietano esplicitamente l'utilizzo di software di scansione automatizzati preesistenti, è stato sviluppato un tool proprietario in linguaggio Python (portscnRange.py). L'obiettivo dello script è verificare l'accessibilità delle porte di comunicazione sui dispositivi di rete, accettando in input un indirizzo IP target e un range di porte definito dall'utente.

3.1 Analisi del Codice Sorgente (Core Logic)

Il cuore del programma risiede nella gestione dei socket TCP. A differenza di un approccio "bloccante" che attende indefinitamente una risposta, lo script implementa un timeout e analizza i codici di ritorno (Return Codes) forniti dal sistema operativo per determinare lo stato preciso della porta (Aperta, Chiusa o Filtrata).

Di seguito è riportato lo snippet fondamentale che gestisce la logica di connessione:

```
# Create a TCP socket (IPv4, TCP)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Optional but recommended: avoid blocking forever
sock.settimeout(0.5)

# Attempt TCP connection
# connect_ex() returns an OS error code
status = sock.connect_ex((target, port))

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# status == 0 → OPEN
if status == 0:
    print(f"Port {port}: OPEN")  # Terminal output
    openPorts.append(port)
    log.write(f"[{timestamp}] Port {port:<5} -> OPEN\n")

# status == 111 → CLOSED (ECONNREFUSED)
elif status == 111:
    log.write(f"[{timestamp}] Port {port:<5} -> CLOSED\n")

# status == 110 → FILTERED (timeout)
elif status == 110:
    log.write(f"[{timestamp}] Port {port:<5} -> FILTERED (timeout)\n")

# status == 11 → FILTERED / WOULD BLOCK
elif status == 11:
    log.write(f"[{timestamp}] Port {port:<5} -> FILTERED (would block)\n")

# Any other error
else:
    log.write(f"[{timestamp}] Port {port:<5} -> ERROR ({status})\n")

# Always close the socket
sock.close()
```

Questa logica garantisce che l'output soddisfi il requisito di fornire una lista delle porte con il relativo stato.

3.2 Proof of Concept (PoC) ed Esecuzione

Per validare il funzionamento dello scanner, è stato eseguito un test contro il Web Server aziendale simulato (macchina Metasploitable), noto per esporre servizi critici.

Dettagli del Test:

- **Script eseguito:** portscnRange.py
- **Target:** 192.168.50.101 (Indirizzo IP del Web Server Target)
- **Range Scansionato:** 10-100 (Include servizi comuni come FTP, SSH, Telnet, HTTP)

```
┌──(kali㉿kali)-[~/Desktop/py2-prog]
└─$ python portscn_02.py
IP to scan: 192.168.50.101
Ports range (Es. 20-100): 10-100
Scanning IP 192.168.50.101 from port 10 to 100
Port 21: OPEN
Port 22: OPEN
Port 23: OPEN
Port 25: OPEN
Port 53: OPEN
Port 80: OPEN

Full scan log saved to: scan_192.168.50.101_10-100.log
```

scan_192.168.50.101_10-100.log

```
1   Port scan results for 192.168.50.101
2   Port range: 10-100
3   Started at: 2025-12-17 08:48:27.465714
4   ----------------------------------------
5   [2025-12-17 08:48:27] Port 10    -> CLOSED
6   [2025-12-17 08:48:27] Port 11    -> CLOSED
7   [2025-12-17 08:48:27] Port 12    -> CLOSED
8   [2025-12-17 08:48:27] Port 21    -> OPEN
9   [2025-12-17 08:48:27] Port 22    -> OPEN
10  [2025-12-17 08:48:27] Port 23    -> OPEN
11  [2025-12-17 08:48:27] Port 24    -> CLOSED
12  [---------Cut for presentation-----------]
13  [----------------------------------------]
14  [2025-12-17 08:48:27] Port 25    -> OPEN
15  [2025-12-17 08:48:27] Port 53    -> OPEN
16  [2025-12-17 08:48:27] Port 54    -> CLOSED
17  [2025-12-17 08:48:27] Port 79    -> CLOSED
18  [2025-12-17 08:48:27] Port 80    -> OPEN
19  [2025-12-17 08:48:27] Port 81    -> CLOSED
20  [2025-12-17 08:48:27] Port 100   -> CLOSED
21  ----------------------------------------
22  Open ports: [21, 22, 23, 25, 53, 80]
```

3.3 Descrizione e Analisi dei Risultati

Come evidenziato dallo screenshot sopra, lo script ha operato correttamente identificando i servizi attivi nel range specificato.

1. **Input Utente:** Il programma ha correttamente richiesto e parsato l'indirizzo IP e il range di porte.
2. **Rilevamento Porte Aperte:** Sono state rilevate porte aperte standard (es. Porta 80 HTTP, Porta 21 FTP, Porta 22 SSH), confermando che il 3-way handshake è avvenuto con successo (Status 0).
3. **Filtraggio Output:** A terminale vengono mostrate solo le porte "OPEN" per immediata leggibilità, mentre l'analisi completa (incluse le porte chiuse o filtrate che hanno restituito codici 111 o 110) è stata salvata nel file di log generato, garantendo una documentazione granulare per l'analisi forense successiva.
4. **Logging:** lo script produce anche un file di log con l'elenco completo delle porte e lo status risultante.

Questa esecuzione conferma che lo strumento è idoneo per mappare la superficie di attacco dell'infrastruttura Theta.

4. Conclusione

L'implementazione di portscnRange.py e portscanALL.py permette alla squadra di sicurezza di Theta di effettuare audit mirati senza violare le policy che vietano tool automatici esterni. L'uso di connect_ex e la gestione specifica dei codici di errore 111 e 110 forniscono una visione chiara non solo dei servizi disponibili, ma anche della configurazione del firewall perimetrale (distinguendo tra porte chiuse e pacchetti droppati).

Questi strumenti sono pronti per essere utilizzati contro il Web Server (macchina DVWA di Metasploitable) e gli altri dispositivi di rete per completare la fase di verifica richiesta dal progetto.